2012-03

Methods for Trustworthy Design of On-Chip Bus Interconnect for General-Purpose Processors

Elson, Jay F.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

METHODS FOR TRUSTWORTHY DESIGN OF ON-CHIP BUS INTERCONNECT FOR GENERAL-PURPOSE PROCESSORS

by

Jay F. Elson

March 2012

Thesis Advisor:	Ted Huffmire
Second Reader:	J.D. Fulp

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2012	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Methods for Trustworthy Design of On-Chip Bus Interconnect for General-Purpose Processors		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Jay F. Elson			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number: N/A			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

Military electronics rely on commodity processors, many of which are manufactured overseas where the trustworthiness of the foundries is uncertain. This thesis attempts to answer the question of whether common bus protocols in use today differ significantly with respect to security, by conducting an analysis of common integrated circuit bus protocols (Inter-Integrated Circuit [I2C], Advanced Microcontroller Bus Architecture [AMBA], HyperTransport, Wishbone, and CoreConnect) based on the Flaw Hypothesis Methodology (FHM). This thesis follows the four stages of FHM. The first stage is Flaw Generation, which involves creating hypothetical attack scenarios. The next is Flaw Confirmation, which involves confirming the flaws generated in the first stage through analysis of the specifications of the bus architectures as well as testing and research in the literature. The third stage is Flaw Generalization, which evaluates the impact of each flaw to determine whether it suggests that a more serious flaw exists in that bus architecture. The final stage is Flaw Elimination, which identifies strategies (and their costs) for mitigating the vulnerabilities based on techniques in the hardware security literature. We conclude that the bus architectures we analyzed differ significantly with respect to security.

14. SUBJECT TERMS IC Bus, Hardware Security, I2C, AMBA, CoreConnect, HyperTransport, Wishbone, Flaw Hypothesis Methodology, Penetration Testing, Flaw Generation, Flaw Confirmation, Flaw Generalization, Flaw Elimination			**15. NUMBER OF PAGES** 113
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

METHODS FOR TRUSTWORTHY DESIGN OF ON-CHIP BUS INTERCONNECT FOR GENERAL PURPOSE PROCESSORS

Jay F. Elson
Lieutenant Commander, United States Navy
B.S., Auburn University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2012

Author: Jay F. Elson

Approved by: Ted Huffmire
Thesis Advisor

J.D. Fulp
Second Reader

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Military electronics rely on commodity processors, many of which are manufactured overseas where the trustworthiness of the foundries is uncertain. This thesis attempts to answer the question of whether common bus protocols in use today differ significantly with respect to security, by conducting an analysis of common integrated circuit bus protocols (Inter-Integrated Circuit [I2C], Advanced Microcontroller Bus Architecture [AMBA], HyperTransport, Wishbone, and CoreConnect) based on the Flaw Hypothesis Methodology (FHM). This thesis follows the four stages of FHM. The first stage is Flaw Generation, which involves creating hypothetical attack scenarios. The next is Flaw Confirmation, which involves confirming the flaws generated in the first stage through analysis of the specifications of the bus architectures as well as testing and research in the literature. The third stage is Flaw Generalization, which evaluates the impact of each flaw to determine whether it suggests that a more serious flaw exists in that bus architecture. The final stage is Flaw Elimination, which identifies strategies (and their costs) for mitigating the vulnerabilities based on techniques in the hardware security literature. We conclude that the bus architectures we analyzed differ significantly with respect to security.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
APB	Advanced Peripheral Bus
APU	Address Protection Unit
ARM	Advanced RISC Machines
ASB	Advanced System Bus
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CAD	Command/Address/Data
CLK	Clock
CTL	Control
CPU	Central Processing Unit
DCR	Device Control Register
DDR	Double Data Rate
DES	Data Encryption Standard
DIFT	Dynamic Information Flow Tracking
DMA	Direct Memory Access
DoS	Denial of Service
DPU	Data-based Protection Unit
FHM	Flaw Hypothesis Methodology
FPGA	Field Programmable Gate Array
Gb/s	Gigabyte/second
GB	Gigabyte
GLIFT	Gate Level Information Flow Tracking
HDL	Hardware Description Language
HT	HyperTransport
I2C	Inter-Integrated Circuit
IC	Integrated Circuit

I/O	Input/Output
IP	Internet Protocol
kbit/s	kilobit/second
KB	Kilobyte
MD5	Message Digest 5
MITM	Man In The Middle
ms	millisecond
NIST	National Institute of Standards and Technology
NoC	Network on Chip
NONSEQ	Nonsequential
NSA	National Security Agency
OPB	On-Chip Peripheral Bus
OS	Operating System
P2P	Point to Point
PDA	Personal Digital Assistants
PE-ICE	Parallelized Encryption and Integrity Checking Engine
PLB	Processor Local Bus
RAM	Random Access Memory
RMW	Read-Modify-Write
ROM	Read Only Memory
RSA	Rivest, Shamir, & Adleman
SCL	Serial Clock Line
SDA	Serial Data Line
SDRAM	Synchronous Dynamic Random Access Memory
SECA	Security Enhanced Communication Architecture
SEI	Security Enforcement Interface
SEM	Security Enforcement Module
SEQ	Sequential
SoC	System-on-Chip
SPU	Sequence-based Protection Unit
SSN	Social Security Number
TCB	Trusted Computing Base
USB	Universal Serial Bus

ACKNOWLEDGMENTS

There are several people I wish to thank. First, my thesis advisor, Professor Ted Huffmire, and my second reader, John D. Fulp, for their guidance, patience, and enthusiasm, which were instrumental in helping me complete my thesis. I also wish to thank Professor Tim Levin for providing a tremendous amount of time, technical information, and knowledge, as well as guidance in proper research procedures. Thank you all for giving me the chance to work in a field of study that was new to me and helping me understand difficult concepts.

To my parents, Steve and Roseanne, thank you for raising me to be the person that I am today and teaching me the meaning of hard work, dedication, and believing in myself.

Finally, and most importantly, I wish to thank my wife, Diana, for her love, support, and patience throughout this entire process. Without her, none of this would have ever been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Military electronics rely on commodity processors, many of which are manufactured overseas. Their trustworthiness is uncertain because there are many opportunities for chips to be compromised at different links of the supply chain. These compromised parts present serious problems for the confidentiality, integrity, and availability of data. Since the hardware is the lowest level of system abstraction, compromised hardware affects the trustworthiness of higher layers that depend on it. Malicious functions implemented at the hardware level have direct control over the lowest level of the system, bypassing intermediate Operating System (OS) layers. For both attacker and defender, there are both opportunities and challenges involved with working at the hardware level. In addition to direct control and bypassing the OS layers, opportunities include the high performance of raw hardware and the possibility of strong physical separation. Challenges include the semantic gap between the hardware level of abstraction and high-level application software, as well as additional engineering and fabrication costs associated with designing and manufacturing custom hardware.

Before addressing detection and mitigation of fabrication design flaws, it is necessary to start with a secure hardware design. Like software, attacks against hardware include both operational and developmental attacks. Operational attacks include differential power analysis, fault injection, and probing. Developmental attacks include malicious design tools and malicious intellectual property. A secure design requires a well-developed threat model, a security policy for addressing the threats, and a system implementation that obeys the policy.

Hardware security is integral to every part of the system such as memory, chipsets, storage devices, controller devices, display devices, motherboards, and bus interconnects. The study and understanding of on-chip bus security is a key component of hardware security because the bus is an essential link for the entire system. No matter how secure other hardware devices are, if the bus is insecure then the risk of data

compromise is high. Hardware security requires careful management of the hardware and software, such that system components obey the overall system security policy.

B. PURPOSE OF STUDY

In this thesis, we investigate methods for trustworthy design and use of on-chip interconnects for general-purpose processors. We perform a vulnerability analysis based on a flaw hypothesis methodology (FHM) [1] of common bus architectures found in general-purpose processors and systems based on general-purpose processors. These architectures include Inter-Integrated Circuit (I2C), Advanced Microcontroller Bus Architecture (AMBA), HyperTransport, Wishbone, and CoreConnect. Results of the vulnerability analysis guide our recommendation of techniques to address bus security problems at the application and the hardware design levels.

C. RESEARCH QUESTIONS

The primary research question of this thesis is: do common buses in use today differ significantly with respect to security? Subsidiary research questions include: How are general-purpose processors susceptible to intentional and unintentional security vulnerabilities in the hardware design? How should engineers design common bus architectures to prevent, detect, and respond to unintentional and intentional security flaws in hardware? What are the current practices and future plans in place to ensure trustworthy designs?

D. RESEARCH HYPOTHESIS

We hypothesize that common buses in use today differ significantly with respect to security, as well as some bus architectures having design features that make them more vulnerable to specific attacks. In addition, we hypothesize that some bus architectures have features that make it easier to incorporate countermeasures. Examples of possible countermeasures include separation of untrusted and trusted components, encryption methods, and sound security policies.

We hypothesize that common bus architectures do not prevent, detect, and respond to unintentional and intentional security flaws in hardware. Buses merely provide

a reliable service of timely data delivery, which ensures availability with respect to security. For those buses that have countermeasures against security attacks, we believe that the costs of those countermeasures are high and that further design improvements are necessary to reduce their performance impact to an acceptable level. We hypothesize that one reason major chip manufacturers are moving to Network-on-Chip (NoC)-based architectures is that a network-based architecture gives greater control over security policy enforcement.

E. ORGANIZATION OF STUDY

The space of hardware security mechanisms for buses and interconnects is large. This thesis focuses on common bus architectures, specifically multicore, on-chip interconnect. The first phase of our study addresses the design specifications of a set of widely used, open bus architectures (AMBA, Wishbone, HyperTransport, etc.). The second phase of our study performs a vulnerability analysis of these bus architectures based on their design specifications. Specifically, how are each of the bus architectures susceptible to attacks that compromise confidentiality, integrity, and availability? The third phase of our study recommends techniques for addressing these vulnerabilities.

Our methodology involves rigorous analysis of the published specifications of a set of open-source bus architectures commonly found in general-purpose processors, with respect to a policy and threat model. We analyze the specifications of the bus architectures in order to determine their vulnerabilities. Once these vulnerabilities are identified (e.g., problems with the design of a bus arbiter), we recommend methods for their mitigation, including prevention, detection, and response. We formulate a precise threat model, security model, concept of operation, semantics of operation, and design specification for a worked example. We express our design example at the block diagram level of abstraction. We leave the transformation of the high-level design to lower levels of refinement to future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INTRODUCTION

Integrated Circuit (IC) bus protocol analysis allows chip designers and computer security professionals to understand, identify, and mitigate security vulnerabilities with respect to confidentiality, integrity, and availability.

To better understand the bus protocols that this thesis investigates, this section provides background on IC bus architecture concepts that includes bus communication components, System-on-Chip (SoC) designs, security policies, and threat models.

B. CORE CONCEPTS OF IC BUS COMMUNICATIONS

Buses are defined as a set of wires that acts as a shared, but common, data path to connect multiple subsystems within a system. Buses are frequently used to transmit information from one component to another and their architectures are based on one of two forms of information flow—parallel or serial form. Parallel buses carry data on multiple wires (several bits of data are sent at the same time along multiple paths), whereas serial buses carry data in a bit serial form (bits of data are sent one at a time along a single path). These design choices are based on platform requirements and the designers' needs [1].

Data is transferred from one device to another on a system. Each device is assigned a role of master, slave, or sometimes both, depending on the platform and particular bus architecture. A master controls the data traffic on the bus. It is responsible for initiating a session by making a read or write request to a device that is designated as a slave. Inversely, a slave performs a service at the behest of a master device. A practical example of a master and slave interaction is if main memory (master device) wanted to write data to a display device (slave).

For data on a bus to travel from one device to another, there has to be a predefined path known as a data line. A data line is a path for the transmission of data between devices. The master and slave devices connected to the bus data line transfer device

addresses, instructions, read/write data, and acknowledgements across the data path. A data path can be unidirectional or bidirectional based on the bus architecture. However, modern bus architectures tend to favor bidirectional in order to save hardware space.

Clock signals are used to inform master and slave devices when to start, stop, and what speed the data is traveling at. A clock line is a path for which systems' clock signals oscillate between high (logical one) and low (logical zero) states. The clock is responsible for speed, bit width, and coordinated actions of a circuit during a bus operation. A clock generator produces clock signals, which synchronize data movement on the data line.

Through any interaction between a master and slave, there has to be assurance that there is no loss of information through bus collisions, deadlocks, or unauthorized use of devices (i.e., depending on the platform, some devices are not permitted to interact with other devices). Every bus architecture has some form of arbitration to ensure that these policies are enforced. Arbitration is the process of determining which bus master will and can obtain access to the bus (e.g., to prevent more than one master from transmitting simultaneously to one slave). Methods of arbitration fall into the categories of either centralized or distributed. A centralized scheme uses a single hardware device, often called an arbiter, which is responsible for allocating time on the bus. The arbiter may be a separate device or part of the processor. In a distributed scheme, there is no central controller. Arbitration is managed through access control logic and all modules act together to share bus resources [1].

Platform architectures are complicated because different devices often have different power and speed requirements (e.g., a processor requires more power than a keypad), there is hardly a one-size-fits-all bus for platforms. A bridge device allows for more than one type of bus architecture to connect to another bus architecture by forwarding data from one bus to another when required. A bridge is able to accomplish this by converting transferred signs in order to satisfy different bus performance standards and protocols.

Peripheral buses are commonly used to connect to bridges. A peripheral bus, also referred to as an "input/output bus" is a data pathway that connects peripheral devices to

the Central Processing Unit CPU. Peripheral buses are generally low speed, have low power requirements, and are less complex compared to other bus architectures. Types of devices that reside on peripheral buses are keypads, timers, and monitors.

The majority of the bus architectures that will be examined in this thesis are SoC buses. A SoC effectively integrates all the components in a computer onto a single IC chip, including all of the necessary components to form a complete system. A SoC consists of microcontrollers or microprocessors, memory blocks, timing sources, peripherals, external interfaces (e.g., Universal Serial Bus [USB] or Ethernet), analog interfaces, and voltage regulators. SoC bus architectures have become more popular because SoC designs usually consume less power and have lower costs and higher reliability than the multichip systems that they have replaced. SoCs also have few packages in the system, which has driven down the assembly cost [2].

Other buses examined in thesis are those that service multicore processors. A multicore processor is a single integrated circuit with two or more independent processors known as "cores," residing together on the same die, which executes instructions. Putting multiple CPUs onto a single die significantly improves performance of cache or bus operations because the signals between devices travel a shorter distance; therefore, the signal is less degraded and propagates more quickly. Higher quality signals allow the transmission of more data in a single clock cycle, and the shorter distances reduces the time it takes to conduct an operation [3].

C. PLATFORM SECURITY

Designers expect that buses obey and support the platform's security policies onto which they are implemented. A security policy defines what it means for a system to be secure. Following National Institute of Standards and Technology (NIST) guidelines, a security policy specifies what actions in the system are legal (e.g., access control). Three distinct faces of data protection are examined in this thesis: confidentiality, integrity, and availability [4].

Confidentiality is concerned with preventing the unauthorized disclosure of information, whereas integrity is concerned with preventing improper modification or

destruction of information. Availability is concerned with timely, reliable access to the data and services to authorized users. Of the three, a violation of integrity would be considered the most severe, followed by confidentiality and availability. However, buses are designed with availability in mind, since they are used to deliver data to the correct device in a timely and reliable manner.

For designers and manufactures of platform systems to ensure that their systems' behavior is according to policy, threat models are used to design test platforms. Threat models consider the question: against what is the system secure? Systems have assets of value that are worth protecting. Threat models help the system designer assess the probability, potential harm, and priority of attacks against system assets in order to minimize those threats. The three general approaches to threat modeling are Attacker-Centric, Designer-Centric, and Asset-Centric [5].

The Attacker-Centric approach to threat modeling involves evaluating the attacker's goals and how the attacker might achieve them. The attacker's motivation could be, for example, to read National Security Agency (NSA) e-mails. A Designer-Centric approach to threat modeling involves focusing on the design of the system and attempting to step through a model of the system, looking for types of attacks against each element of the model. The Asset-Centric approach considers the assets entrusted to a system (e.g., customer database) and what can be achieved by exploiting them, such as sensitive personal information (e.g., Social Security Number [SSN], credit card numbers, etc.).

For future analysis and discovery of bus flaws in this thesis, we will focus on all three threat models where applicable, but more so on Design-Centric. We are concerned about an attacker discovering vulnerabilities in hardware systems by analyzing, for example, computer buses, SoCs, and multicore processors.

D. RELATED WORK

Attacks on computer hardware have increased over the past few years as computers have become more advanced and integrated into our personal and professional lives. As attackers become increasingly skilled, significant effort has been undertaken to

combat this emerging threat. This thesis only examines security vulnerabilities for on-chip buses, but there is a wide range of security attacks and vulnerabilities on other aspects of computer hardware.

Less than a decade ago, attacks were largely software-related, such as buffer overflows, worms, viruses, etc. Lately, however, attacks have focused on OSs, device drivers, chipsets, memory, and peripherals. Attackers look to exploit hardware because it is the most privileged entity and is able to give the attacker considerable flexibility and power.

Four types of hardware attacks are possible. In the first type, an adversary actively manipulates the control signals of a platform in order to subvert the platform. This requires that an attacker have physical access to the platform. In the second type, an attacker looks for vulnerability in the interaction between two or more components and exploits the vulnerability. In the third type, an attacker looks for vulnerabilities in boot-up/initialization configurations in order to launch hardware attacks. In the last type, an untrusted or less privileged user influences hardware operations. An adversary compromises platform security by exploiting an untrusted component on a platform in order to maliciously influence the operation of the hardware [6].

Hardware Trojans, also known as malicious inclusions, may compromise confidentiality, integrity, or availability (e.g., by causing unauthorized disclosure or modification of data) [7]. In some cases, the compromise of hardware destined for military systems occurs at the foundry. In 2008, Israeli jets bombed a suspected nuclear installation in Syria. The Syrian radar was state-of-the-art and was supposed to warn for impending attacks, but it failed to do so. Experts have speculated that subversion of the commercial off-the-shelf microprocessors in the Syrian radar occurred during fabrication, allowing the Israelis to block the radar [8]. The design space for malicious circuitry is large, and attackers have the advantage of stealth when designing attacks. These types of attacks are practical, flexible, and difficult to detect [9].

With a wide range of attacks possible, the job of the security specialist becomes more challenging, and designers must develop and deploy effective countermeasures.

E. SUMMARY

This chapter provided background on critical concepts and terms related to IC buses and hardware architecture. Understanding common terms, current attacks, and future threats for computer hardware platforms is necessary to understand the following chapters, which explore how to identify and analyze risks in order to mitigate attacks.

III. BUS ARCHITECTURES

A. INTRODUCTION

We will discuss and define five common IC bus architectures currently in use and on the market. Understanding bus architectures comes through defining and describing their purpose, design principles, components, protocol specifications, and common use case scenarios. These will serve as a foundation for later chapters.

B. INTER-INTEGRATED CIRCUIT (I2C)

Philips Semiconductors developed I2C as an early implementation of an IC bus in the 1980s to aid communications between components that reside on the same circuit board (peripherals, mother boards, embedded systems, etc.). Today, more than 50 companies worldwide have implemented I2C on over 1,000 different IC boards [10].

I2C buses are attractive for system designers because they are compatible with almost any IC, facilitating rapid prototyping of designs and allowing upgrades and modifications by adding or removing devices. Designers and manufacturers advertise I2C for its low power consumption, wide operating temperature range, and high noise immunity. I2C eliminates the need for address decoders and glue logic, and it reduces space requirements, which keeps designs simple and flexible. It also supports simple constructions and enables easy upgrades. I2C buses are popular in the marketplace for low-speed peripheral devices such as radios, televisions, and personal digital assistants (PDA).

I2C has a physical layout of two bidirectional wires, Serial Data Line (SDA) and Serial Clock Line (SCL), which transmit information between devices. Each device connected to the bus has a unique address assigned to it and can operate in receive and/or transmit mode with a designation as a master or slave. I2C offers the possibility of having multiple masters; however, only one master can transmit data over the bus at a time. If there is an instance of two masters simultaneously transmitting data on the bus, then arbitration procedures resolve the contention. For example, when the two competing masters begin to write simultaneously to the bus, both masters will begin to write to the

bus. The first master who writes a “one” loses the contest and ceases its operation, while the other master continues with its intended operation.

Figure 1 exhibits the topology of I2C. The SDA and SCL are bidirectional, and when the bus is available, both lines are HIGH. When the SDA line is HIGH, it represents a logical one, and when the SDA is LOW, it represents a logical zero. Data transfers on the bus occur at a range of speeds from 100 kilobits/second (kbit/s) to 400 kbit/s. The data transferred across the SDA line must be stable during HIGH periods of the clock. The state of the SDA can only change during LOW periods of the SCL. One clock pulse is generated for each data bit transferred. A Start condition is generated on the bus when there is a HIGH to LOW transition on the SDA line while the SCL is HIGH. A Stop condition is defined as a LOW to HIGH transition of the SDA line while the SCL is HIGH. Masters are the only devices that may initiate START and STOP conditions. The bus is busy when there is a START condition, but free when there is a STOP condition. The bus will remain busy if the master initiates another START condition vice a STOP condition. Figure 2 depicts high and low states that initiate and terminate transmissions on the bus.

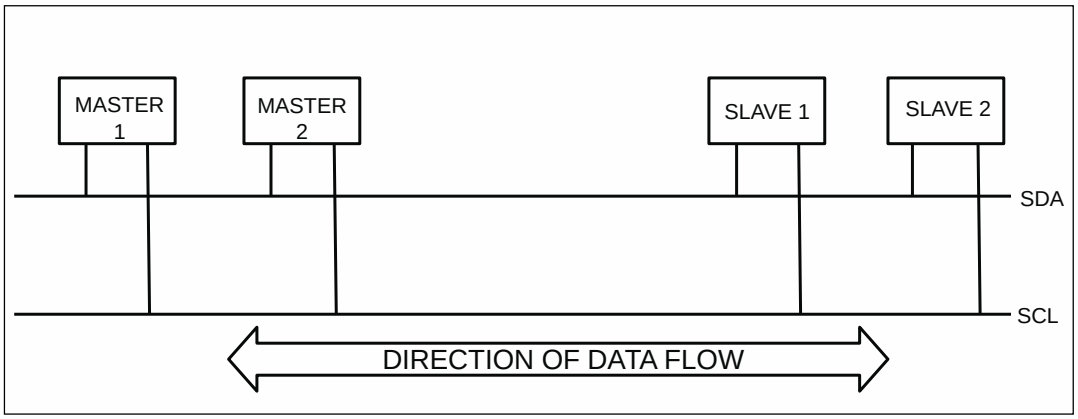

Figure 1. I2C bus topology (From [10]).

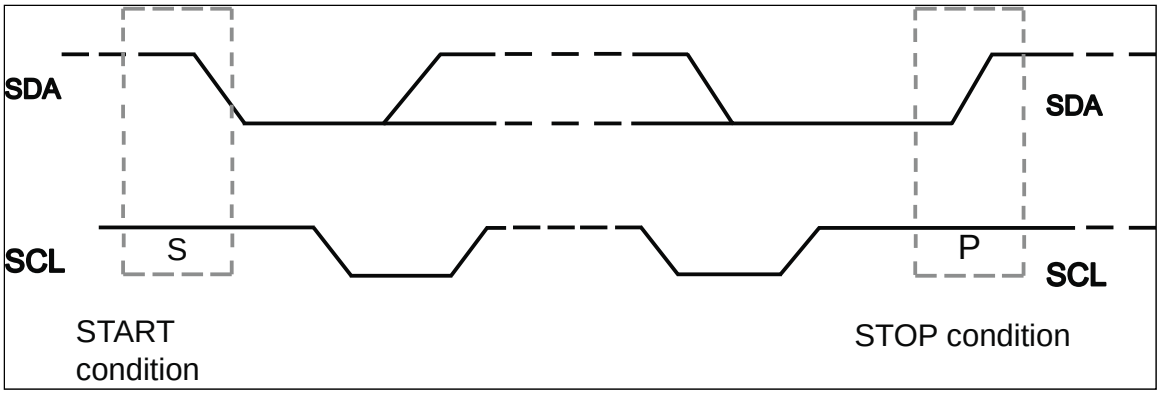

Figure 2. I2C START and STOP conditions (From [10]).

I2C requires each byte of data to be eight bits in length before it is placed on the SDA line. There is no restriction on the amount of data that can be transmitted per transfer. Slave devices have the option to force the master device into a wait status by lowering the SCL from HIGH to LOW whenever it cannot receive/transmit a complete byte or to force an interrupt. The slave device that forced the wait status on the bus is the only device that can end it by raising the SCL from LOW to HIGH. Each byte during a transfer has to be followed by an acknowledgement bit, which signals to the transmitting device that the byte was successfully received and that another byte may be sent. Figure 3 depicts an I2C sequence.

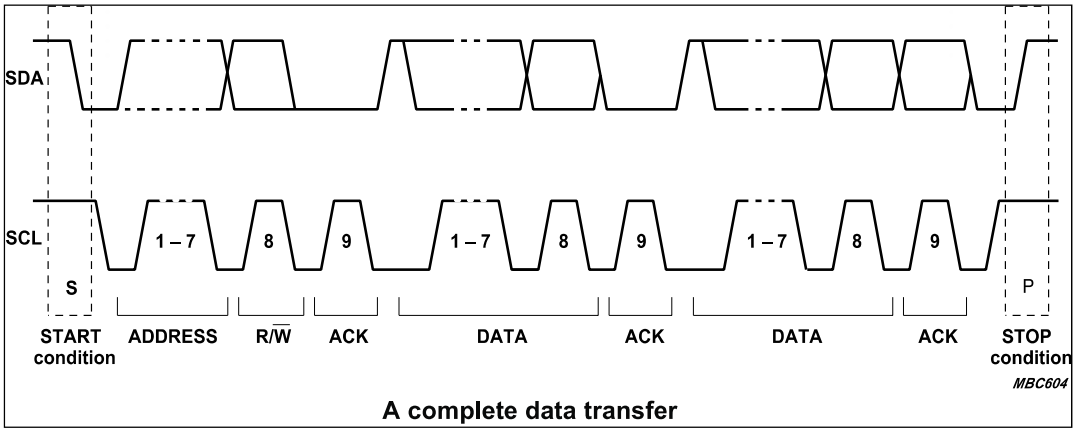

Figure 3. I2C byte format (From [10]).

A typical I2C operation begins with a master addressing a slave to either perform a read or write. The master directly addresses the slave by its unique seven-bit address immediately followed by a bit to represent either a read operation, which corresponds to a zero, or a write operation, which corresponds to a one. The slave will immediately send an acknowledgement followed by the master's sending n bytes of data [10]. The slave will acknowledge receiving the n bytes, and this process of sending n bytes followed by an acknowledgement continues until the master terminates the session. Figures 4 through 10 are from an animation program that the author designed in Adobe Flash CS5 that will help the reader to understand and analyze I2C bus operations.

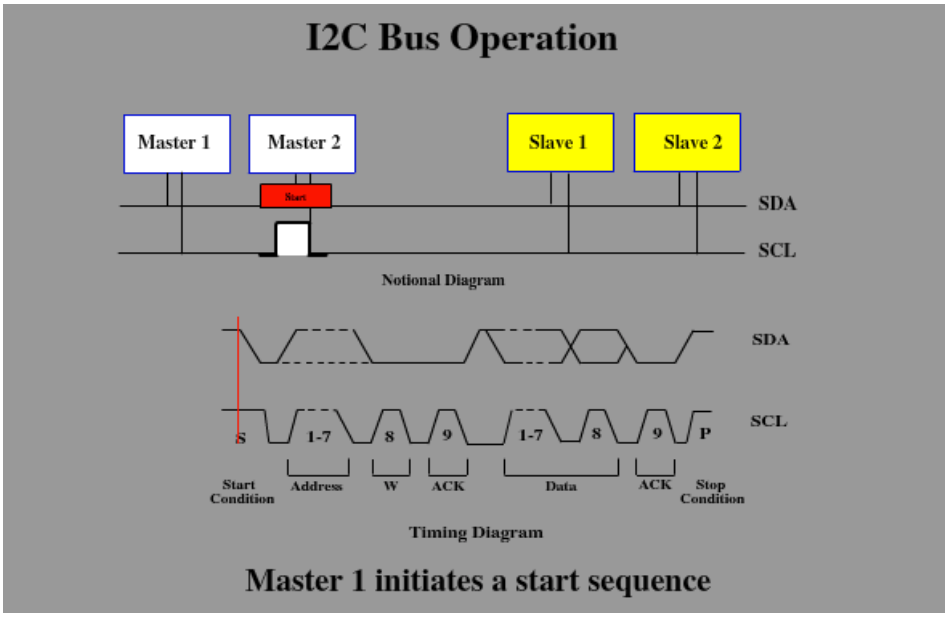

Figure 4. I2C START sequence.

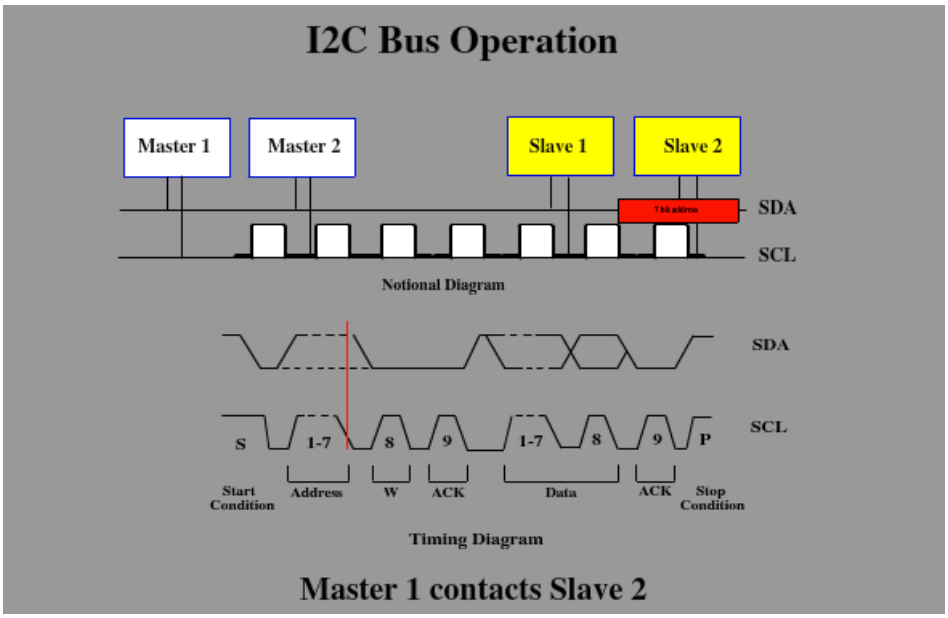

Figure 5. Slave addressing.

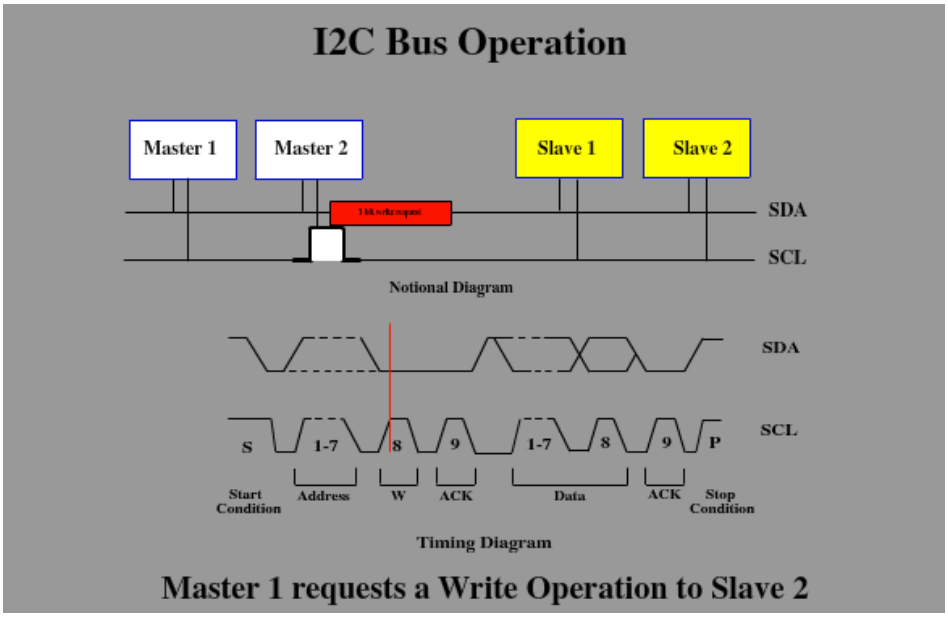

Figure 6. Write operation.

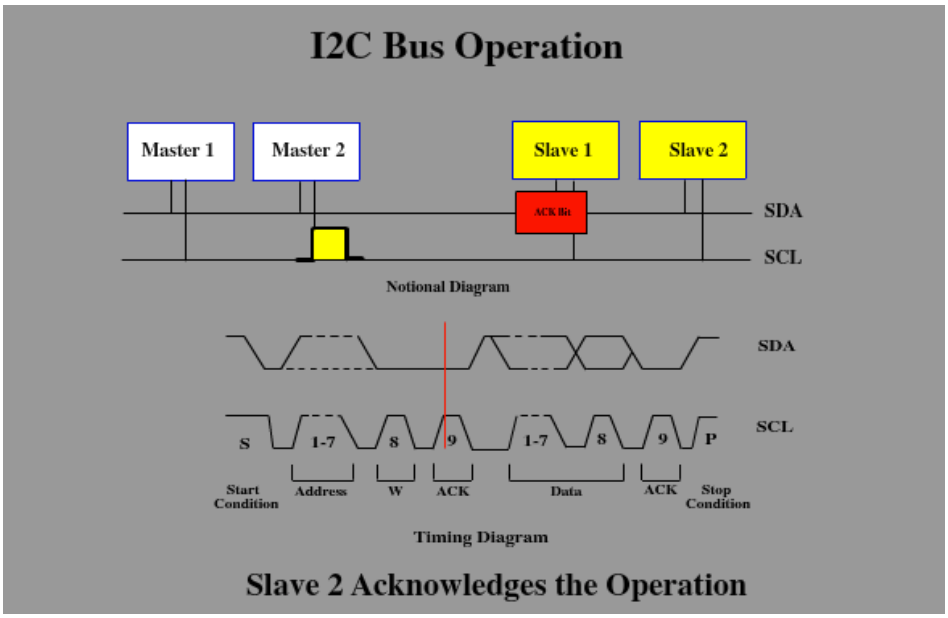

Figure 7. Slave acknowledgement.

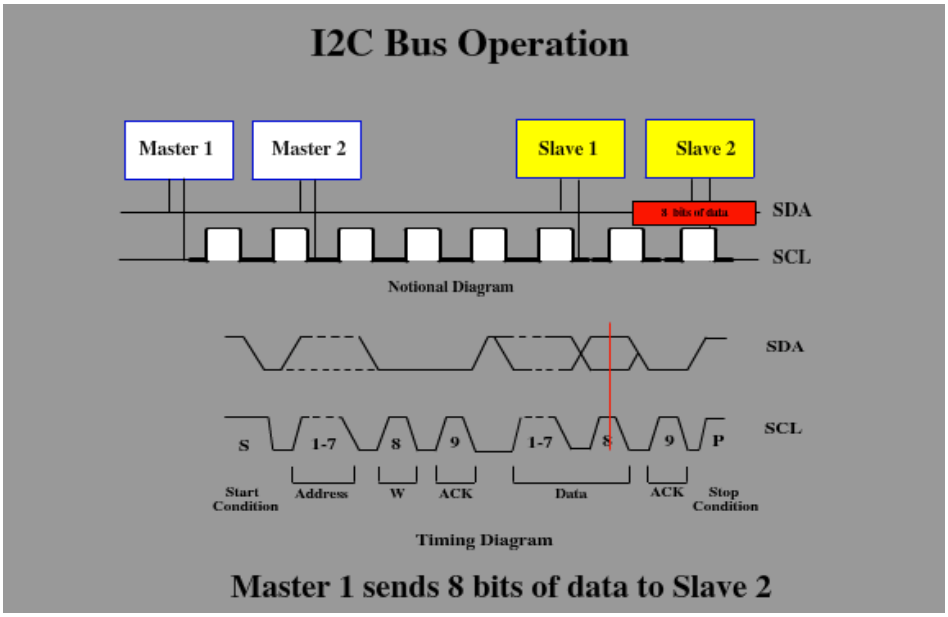

Figure 8. Data transfer.

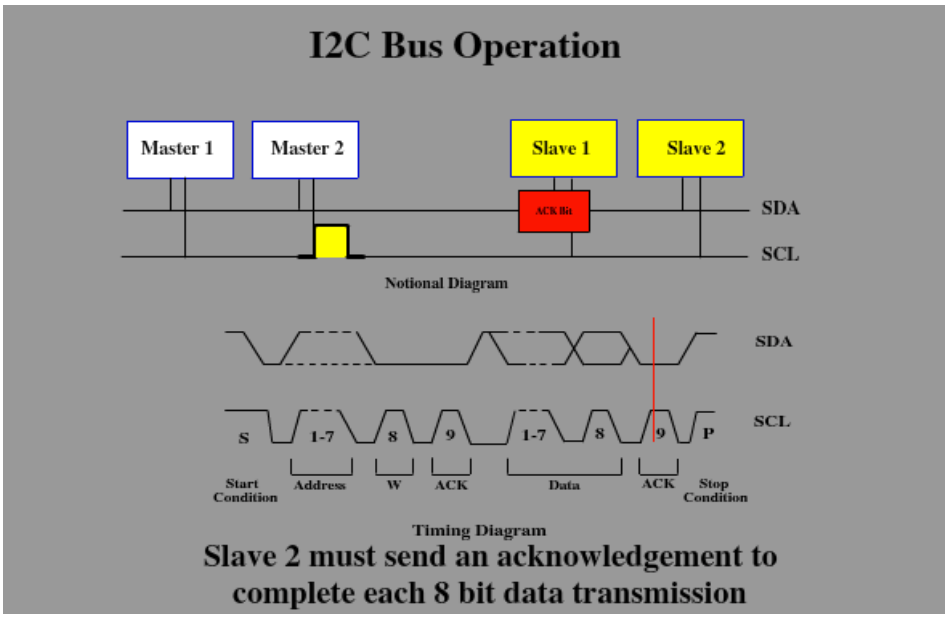

Figure 9. Data acknowledgement.

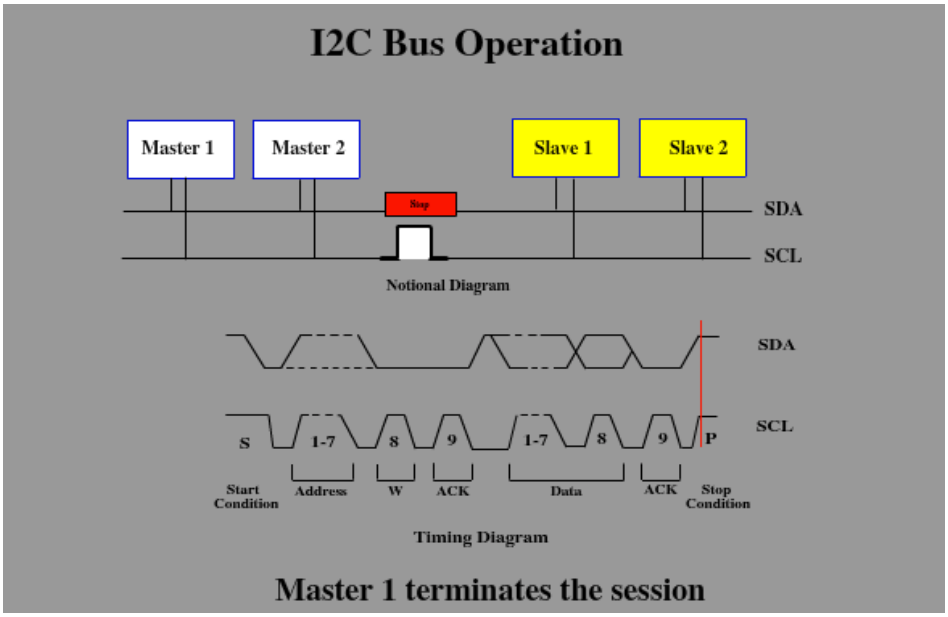

Figure 10. Sequence termination.

C. ADVANCED MICROCONTROLLER BUS ARCHITECTURE (AMBA)

AMBA was developed by ARM Ltd. in the mid-1990s and is widely used in a range of Application Specific Integrated Circuit (ASIC) and SoC parts. It is considered the SoC bus of choice among developers and manufacturers because of its ability for reuse, modularity, and its infrastructure that supports high performance and low power consumption for on-chip communications.

AMBA's design enables reuse of Internet Protocol (IP) cores and IC processes. Manufacturers emphasize modularity to achieve technology independence and maximum performance. AMBA is unique in that is it has many distinctly different specifications, versions, bus types, etc. In this thesis, we will examine AMBA version 3.0, which is made up of three distinct bus architectures. The first is the Advanced High-Performance Bus (AHB), which is used as the backbone for high-performance systems and supports connections between processors, on-chip communications, and off-chip communications. The second type is the Advanced System Bus (ASB), which is a less complex alternative to AHB. The third is the Advanced Peripheral Bus (APB), which is optimized for minimal power consumption and is used for interfacing peripheral devices that do not require high performance or high bandwidth. Figure 11 depicts the standard AMBA version 3.0 topology [11].

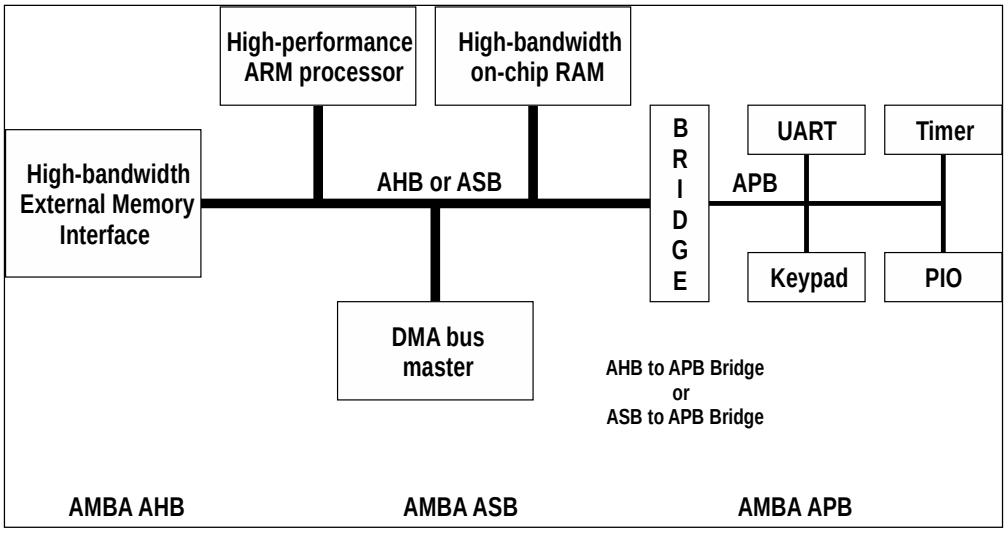

Figure 11. AMBA architecture (From [11]).

AHB is a new generation of the AMBA ASB bus; its purpose is to address the requirements of high-performance designs. AHB features include:

- Burst transfers
- Split transactions
- Single-cycle bus master handover
- Single-clock edge operation
- Multiplexing
- Wider data bus configuration [11]

AHB employs basic and burst transfer sequences. Basic transfers are broken into two phases: address sequence and data transfer. The master devices send out an address signal during the address phase. During the transfer phase, the slave device sends data and awaits the appropriate response. The address phase will only last for one clock cycle. Slave devices can insert wait cycles into transfer operations in order to have more time to prepare valid data. When performing read operations, slave devices assert the ready signal until the read data is valid, but when a write operation is performed, the master device holds the write data until the slave device asserts the ready signal. AHB can also support pipelined operations for the data phase of the first transfer, and the address phase of the second transfer can overlap to improve bus performance.

Burst transfers are sequential transfers that are written to the same memory space. Four types of burst transfers are IDLE, BUSY, NONSEQ (nonsequential), and SEQ (sequential).

IDLE indicates that no data transfer is required. BUSY allows the master to insert idle cycles during a burst transfer because the master device wants to continue with a burst transfer, but the next transfer cannot take place immediately. NONSEQ indicates that the first transfer of a burst transfer is a single transfer. SEQ indicates that the address is related to the previous transfer. Burst transfers cannot exceed the 1 KiloByte (KB) address boundary. Aside from transfer and burst types, each transfer will have a number of control signals that provide additional information about the transfer.

The central address decoder is used to provide select signals for each slave on the bus. The select signal is a combination of high order address signals and a simple address bus decoding scheme to avoid complex logic and ensure high-speed operation. The slave must sample the address and control signals, and when HREADY is HIGH, the slave must indicate the current transfer.

After a master has initiated a transfer, the slave then determines how the transfer should progress. Similar to an I2C slave device, the slave can complete the transfer immediately, pose an interrupt/wait state, or signal an error to indicate a failure to transfer. However, unlike an I2C slave, the slave device can delay the completion of a transfer and allow both master and slave to back off the bus, leaving the bus available for other transfers.

Transfer responses are OKAY, ERROR, RETRY, and SPLIT. OKAY indicates that the transfer has successfully completed. An ERROR response indicates that some form of error has incurred and notifies the master that the transfer was unsuccessful. RETRY indicates that the transfer has not yet completed, so the bus master should reinitiate transfers until it is complete, but a two-cycle RETRY response is required. SPLIT indicates that the transfer has not yet completed, but the bus master must retry the transfer the next time it may access the bus. However, a two-cycle SPLIT response is required.

Arbitration ensures that only one master has access to the bus at a time. The central arbiter receives address and control signals from the bus master and then determines which device has the highest priority. In addition, the arbiter will manage requests from slave devices during SPLIT transfers.

SPLIT transfers improve the utilization of the bus by separating (1) the operation of the master and providing the address to the slave from (2) the operation of the slave responding to the appropriate data. A SPLIT transfer occurs when a slave determines that a transfer will take a large number of clock cycles to complete.

Figure 12 shows a standard ABH interconnection for a standard bus sequence. A typical operational scenario of AHB would involve a master requesting access to a slave

to perform a write operation. The arbiter will receive the request signal and determine whether the requesting master device has permission to access the slave device and whether the slave is available (i.e., not performing another operation). Assuming the master device has the appropriate access and the slave device is free from use, the arbiter then transfers the address and control signals to the slave device. The control signals provide the information, direction, and width of the transfer and indicate whether a burst transfer is required. During the transfer, the slave shows the status using response signals (i.e., OKAY, ERROR, RETRY, and SPLIT).

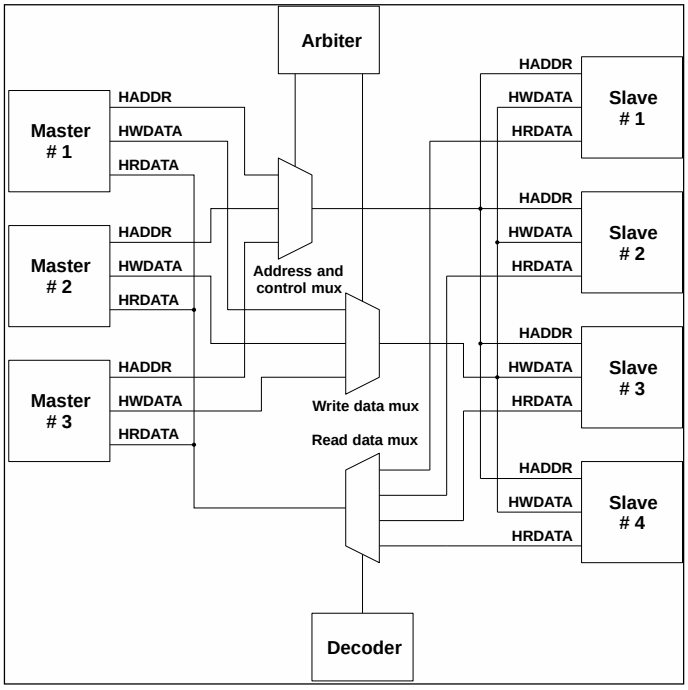

Figure 12. AHB interconnection (From [11]).

ASB is similar to AHB except that it cannot perform SPLIT transactions. Its bus protocol can be used with a central multiplexor interconnection scheme. Using the interconnection scheme, the bus master will send address and control signals to indicate the desired operation to the central arbiter. The central arbiter reviews the bus master's address and control signals and determines whether the bus master has the appropriate access to the desired slave device (i.e., the master may have read access, but no write

access). Data read and response signals from the multiplexor require a central decoder, which will select the appropriate signals from the slave device.

Similar to I2C, the APB is designed for minimal power consumption and reduced complexity. APBs interface with low power, low bandwidth, and low-performance peripherals. The bridge interface between APB and ASB/AHB is the only bus master for APB, but is a slave device on the high-performance ASB/AHB.

An APB slave has a simple and flexible interface. Its exact implementation details depend on individual design requirements. Typical operations of an APB slave connected to an ASB bus are read and write transfers; however, an APB slave interfacing with an AHB performs the same operations as an APB slave connected to an ASB, but also can perform back-to-back transfers and utilize multiplexing data bus implementations. Multiplexing supports combining read and write data buses into a single bus in which read and write operations never occur simultaneously.

D. HYPERTRANSPORT

The HyperTransport Consortium was formed in the early 2000s to develop the HyperTransport (HT) bus that would provide a flexible, reusable, and high-speed IC bus at a low cost. Improvements to CPU execution and other devices have placed increasing demands on external resources like off-chip memory, which are comparatively slow in speed, negatively affecting overall system performance. In July 2007, an AMD analyst discovered that 1 millisecond (ms) of latency was equivalent to $100 million in stock [12].

HT is a high-speed, high-performance, point-to-point interconnect scheme that can support a wide range of devices. HT provides a simple and scalable solution that can provide various applications with bandwidths up to 5.1 Gigabytes (GBs) [12]. The HT architecture comprises a series of devices that form *tunnels*. A tunnel is an HT device that performs some function and has a second interface that permits the connection of another device. The end device is a *cave* that represents the termination of a chain of devices that

connect to the same HT bus. A series of HT buses is an HT chain. Additional HT buses (i.e., chains) may be incorporated into a system by using an HT-to-HT bridge, as seen in Figure 13.

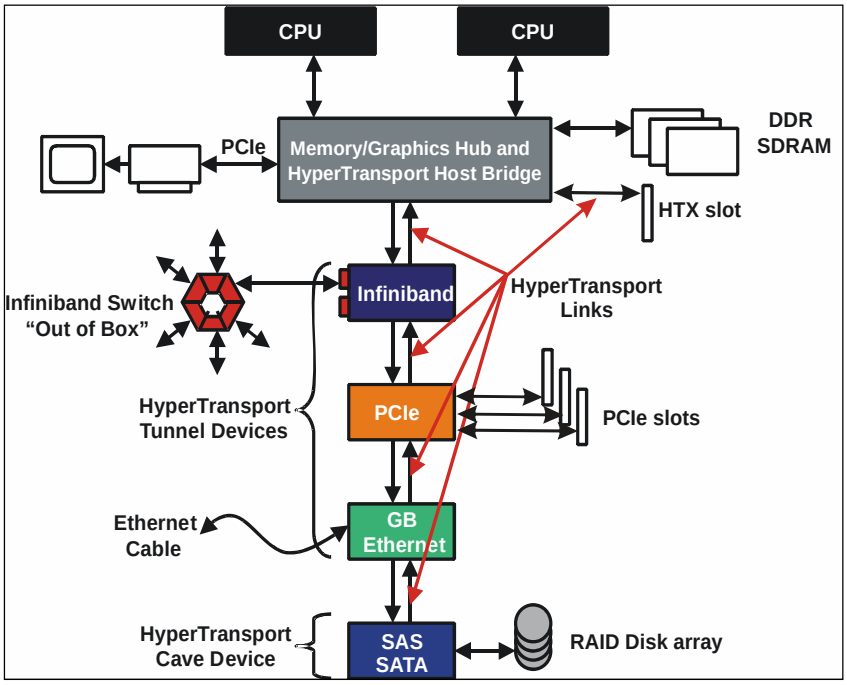

Figure 13. HT architecture overview (From [12]).

HT signals consist of unidirectional signals that are outbound and inbound for each device and are routed from point to point. HT signals fall into two groups: high speed and low speed. High-speed signals are associated with sending and receiving of control and data packets. Low-speed signals are required for reset and power management. High-speed signals consist of Command/Address/Data (CAD), Clock (CLK), and Control (CTL) signals. CAD signals transmit HT packets, which consist of requests, responses, and data. Each CAD bus may consist of 2 to 32 bits (i.e., 2 to 32 different signal pairs). The receiver uses CTL to identify the information sent over the CAD signals. HT uses CLK along with CAD and CTL to transmit. There is one CLK signal pair for each set of eight or fewer CAD signal pairs to aid in the partitioning of on-chip clock circuitry as well as easy board layout.

HT utilizes a packet-based protocol in which all information, addresses, commands, and data travel in packets, which are multiples of four bytes each (i.e., 4, 8, 16, or 32 bits). The number of bytes in an HT packet will determine the length of the bit time for information to travel. Bit time is the length of time it takes a bit to travel at its predefined speed. There are two bit times per clock period. For example, given an 8-bit interface, one byte of packet information is sent one bit at a time.

HT packets have ordering rules to avoid deadlock (e.g., two separate transactions are each dependent on the other completing first), support legacy buses, and maximize performance. HT packets are ordered based on rules. Rules are grouped into the following categories: (1) general rules, (2) rules for upstream Input/Output (I/O), and (3) rules for downstream ordering. Ordering rules apply to the order in which operations are detected by targets.

There are three types of traffic flows, as seen in Figure 14: Programmed I/O traffic, DMA traffic, and Peer-to-Peer traffic. Programmed I/O traffic originates at the host bridge on behalf of the CPU and target or Memory Mapped I/O in one of the peripherals. These transactions are used to evaluate status and program configuration spaces. DMA traffic originates on the bus master peripheral and targets main memory. Traffic is managed so that the CPU may be relieved of the burden of moving large amounts of data to and from the I/O subsystem. Peer-to-Peer traffic is generated by an interior node and targets another interior node.

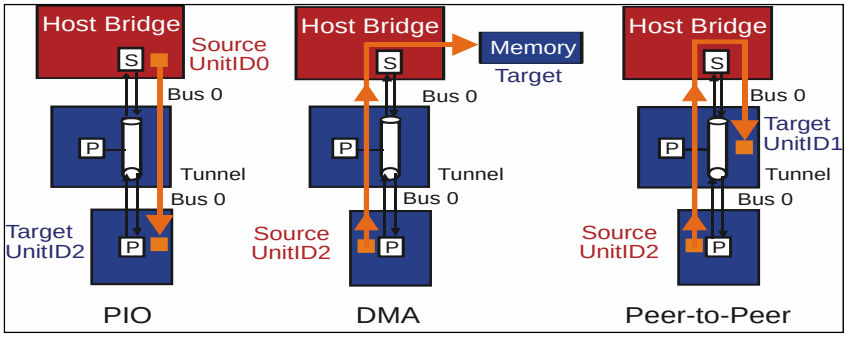

Figure 14. HT traffic flow (From [12]).

In summary, the HT bus protocol has information flowing in a unidirectional manner as it moves from device to device. This is contrary to I2C and AMBA traffic schema. As with a network protocol, information that travels on the HT bus is organized into data packets, with each device checking header information along the way to see if the packet belongs to that particular device. HT technology is high bandwidth, low latency, scalable, and extensible. HT eliminates the need for multiple local buses, resulting in simple design and implementation. HT has provided performance benefits that exceed present industry requirements and will ideally serve future market needs throughout the industry.

E. WISHBONE

Wishbone is a SoC bus for portable IP cores and offers perhaps the greatest flexibility in design methodology with semiconductor IP cores. Wishbone is a product of OpenCores, which is an open-source hardware community for professionals and hardware design enthusiasts. Similar to AMBA, the purpose of Wishbone is to ease the integration of SoC components through design reuse [13].

Wishbone's features include:

- Simple to understand and easy to use
- Flexible and portable in support of object reuse
- Designers have the benefit of determining their own Arbitration schema

Objectives of Wishbone include:

- Flexible interconnection between IP cores
- Encourage design reuse through compatibility of IP cores
- Ensure that the protocol and architecture is easy to understand by the developer and user

Wishbone was given its name because its interface has a separate input and output for each device. Wishbone is intentionally ambiguous because it was intended to let designers use several designs written in Verilog or some other hardware description language (HDL) of their choice.

There are three common architectures associated with Wishbone: Shared Bus, Pipeline, and Crossbar. Designers will choose a shared bus interconnection when there are two or more masters that need to be connected to one or more slaves. The master initiates a bus cycle to a target slave, and then the target slave participates in one or more bus cycles with the master. An arbiter determines when a master may gain access to the shared bus. An arbiter acts like a traffic cop and dictates how shared resources can be accessed. The advantage of this configuration (shown in Figure 15) is that shared interconnection systems are relatively compact, and few resources are required for configuration and logic gate routing. However, it has a disadvantage with respect to the sharing of resources. Master devices potentially have to wait for access to the bus, which degrades the speed and efficiency of access to bus resources.

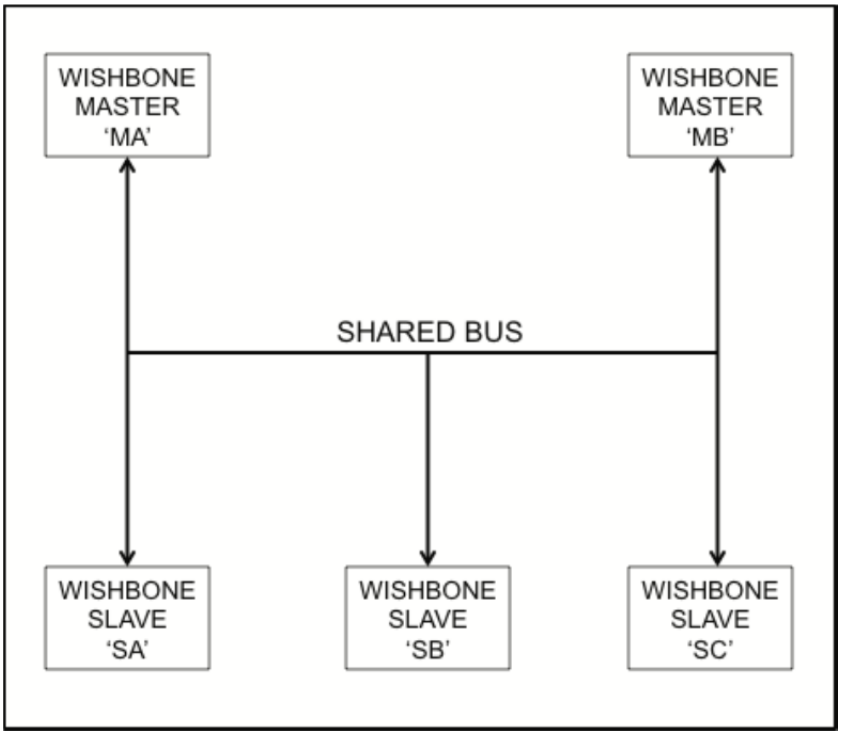

Figure 15. Wishbone shared bus (From [13]).

A crossbar connects two or more masters so that each can access two or more slaves. In this configuration, a master initiates an addressable bus cycle to a target slave. An arbiter determines when each master may gain access to that slave. As seen in

Figure 16, a crossbar switch is more complex than a shared bus, but it offers significant advantages. First, a crossbar switch allows two or masters to access the bus at the same time as long as they are not accessing the same slave device. A crossbar switch also offers a higher data transfer speed. However, a crossbar switch does have some disadvantages compared to a shared bus. For example, a crossbar switch requires more interconnection logic, and it needs more resources to be able to route data.

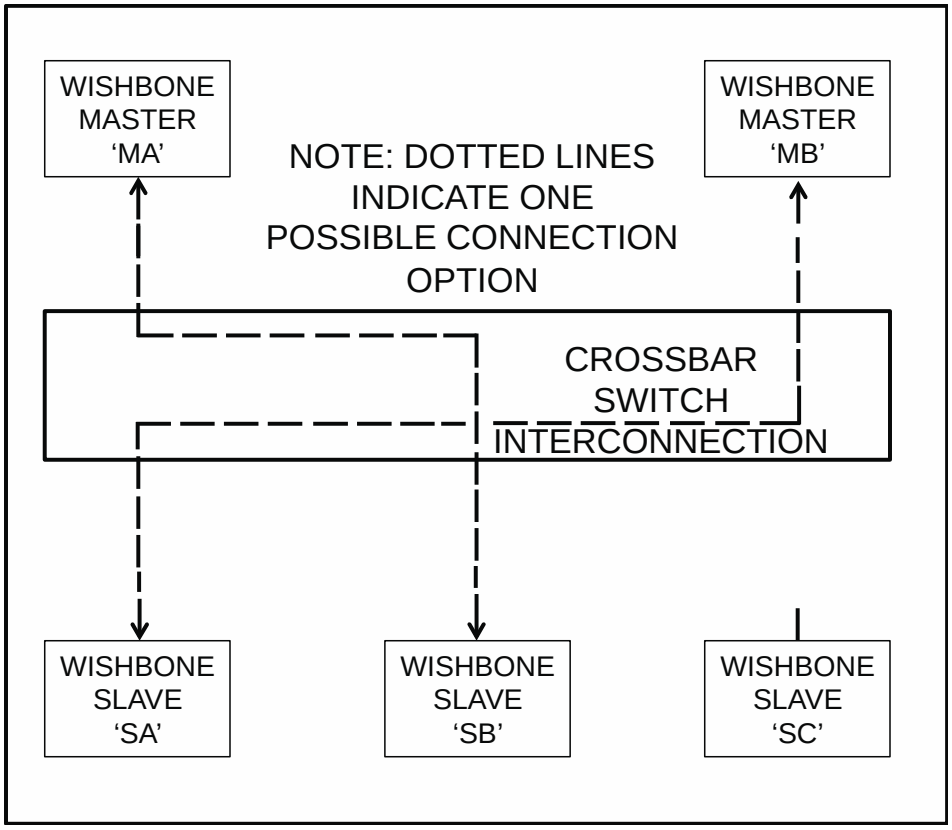

Figure 16. Wishbone crossbar switch (From [13]).

Perhaps the simplest topology is a pipelined topology, as shown in Figure 17, in which data is processed in a sequential manner (similar to an HT architecture). A data flow architecture exploits parallelism, which speeds up execution time. Users of the pipelined architecture assume that every operation will take an equal amount of time and that all operations occur in a sequential manner.

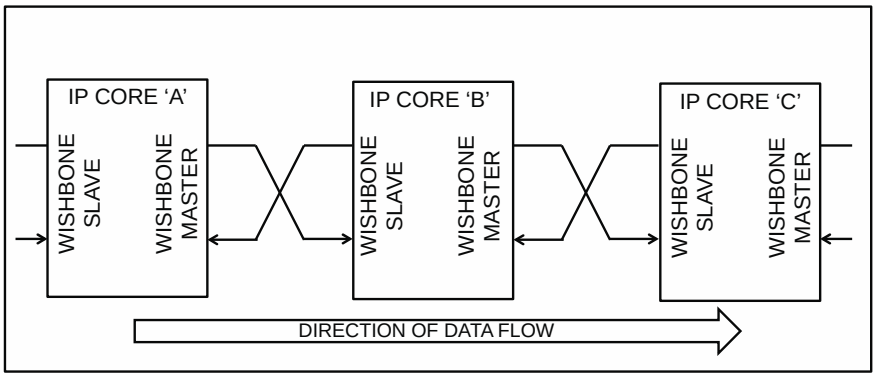

Figure 17. Wishbone pipeline (From [13]).

Wishbone designers are given a lot of freedom in their designs; however, they must have some consistency such as classic bus transfers with respect to read/write operations, read-modify-write (RMW), and resets. In single read/write cycles there is one data transfer at a time. These are the simplest data transfers on Wishbone. For block read/write cycles, the block transfer cycles perform multiple data transfers. An RMW cycle is capable of data transfer every clock cycle, and resets have all hardware interfaces initialized at predefined states when the reset signal is asserted.

Wishbone specifications provide the end user with simple timing constraints. Application-specific circuits will vary, but in all cases, the only timing information needed by the end user is the maximum clock frequency. The maximum clock frequency is measured as the time from the positive edge clock of the input device to the positive edge clock of the next device in the bus path.

Wishbone is an open-source bus designed for hardware design professionals and enthusiasts to alleviate interconnection difficulties and to encourage the sharing of designs. A device cannot be Wishbone compliant unless it includes a data sheet that describes what it does, its bus width, utilization, etc. Wishbone is perhaps the least conventional of all the buses described in this chapter, but it is the easiest to connect to cores and therefore easier to create a SoC bus [13].

F. CORECONNECT

CoreConnect is a SoC bus developed by IBM to be sufficiently flexible and robust in supporting a wide variety of system needs. It directly competes against ARM Ltd.'s AMBA SoC bus in the open market. CoreConnect resembles ARM's AMBA and OpenCores' Wishbone SoC buses in that both are promoted for their flexibility and reuse capabilities. CoreConnect's specification has three bus architectures for interconnecting cores, which have their own library macros and custom logic design. These buses are Processor Local Bus (PLB), On-Chip Peripheral Bus (OPB), and Device Control Register (DCR). Figure 18 illustrates the CoreConnect architecture.

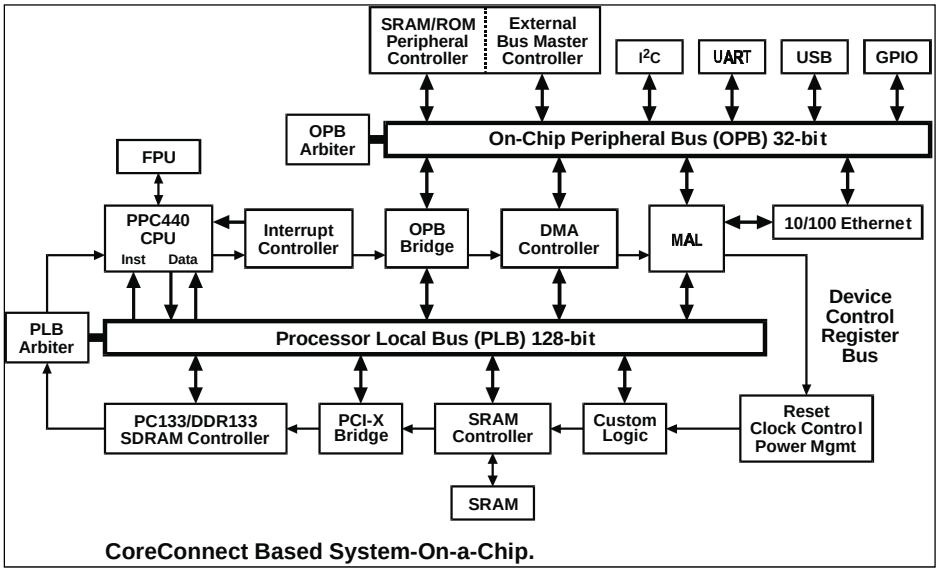

Figure 18. CoreConnect SoC (From [14]).

The CoreConnect and AMBA bus architectures share many similarities. They both have high-performance features and support bus widths of 32 bits and higher. They both have separate read and write paths for greater bus usage and for allowing multiple masters. They both provide high-performance features such as pipelining, split transactions, and burst transfers. CoreConnect and AMBA differ greatly in marketing and design. IBM offers a no-fee, royalty-free architectural license for CoreConnect, unlike ARM, which licenses AMBA. Table 1 compares AMBA 2.0 and CoreConnect [14].

	IBM CoreConnect Processor Local Bus	ARM AMBA 2.0 AMBA High-Performance Bus
Bus Architecture	32, 64, and 128 bits Extendable to 256-bits	32, 64, and 128 bits
Data Buses	Separate Read and Write	Separate Read and Write
Key Capabilities	Multiple Bus Masters 4 Deep Read Pipelining 2 Deep Write Pipelining Split Transactions Bus Transfers Line Transfers	Multiple Bus Masters Pipelining Split Transactions Burst Transfers Line Transfers
	On-Chip Peripheral Bus	**AMBA Advanced Peripheral Bus**
Masters Supported	Supports Multiple Masters	Single Master: The APB Bridge
Bridge Function	Master on PLB or OPB	APB Master Only
Data Buses	Separate Read and Write	Separate or 3-state

Table 1. Comparison of CoreConnect and AMBA 2.0 architecture (From [14]).

The PLB and OPB buses provide the capability of data flow. PLB addresses performance, latency, and design flexibility issues such as:

- Capable of providing split transactions
- Latency reduction via address pipelining
- Concurrent read and write operations
- Bus requests and grant protocols may overlap with ongoing read/write transfers [14]

PLB offers designers flexibility through the following features:

- Completely synchronous operations
- Allowing arbitration to resolve deadlock situations
- Fully synchronous operations
- Master driven operations
- Slave error reporting [14]

Figure 19 shows a typical PLB architecture, where each PLB master is attached to the PLB macro through separate address, read data, and write data buses. PLB slaves are attached to the PLB macro through sharing a decoupled address, read data, and write data bus with status signals for each data bus.

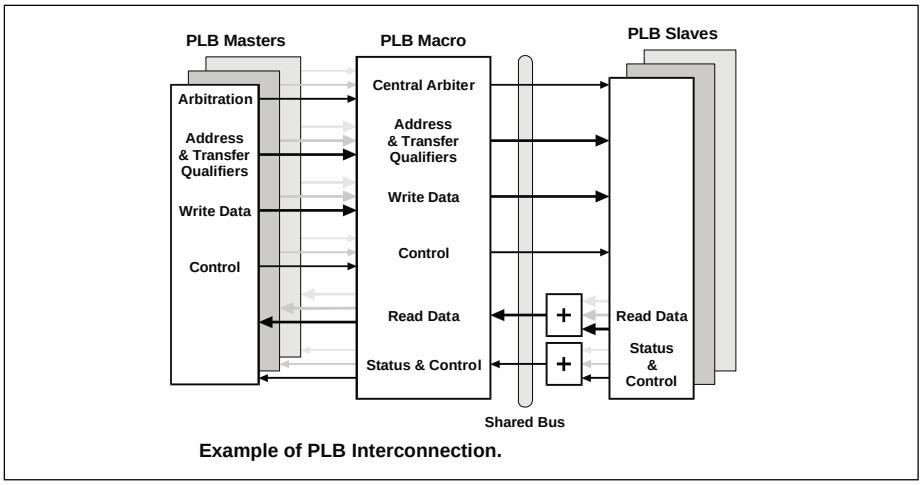

Figure 19. PLB interconnection (From [14]).

A PLB architecture can support up to 16 master devices and any number of slave devices. The number of masters and slaves directly affects the maximum attainable clock rate (i.e., the more devices working on the bus, the slower the bus can be).

PLB consists of a bus arbitration control unit and control logic to manage address and data flow throughout. Master devices are allowed simultaneous address and data transfers. Arbitration manages requests and directs authorized address and data information from a master to a slave device, as well as responses from the slave to the master.

A PLB transaction consists of multiple address and data transfers. Depending on activity, there may be more than one bus cycle to accomplish a transaction. Bus transactions are similar to AMBA in that transactions begin with a master requesting ownership of a slave device by sending data and transfer signals to the arbiter. A PLB arbiter grants ownership, and the master is connected to the slave device to begin the transaction phase. Once the slave responds with an acknowledgment, data flow may begin. The data transfers have two phases: transfer and acknowledgment. During the transfer phase, the master drives write operations and samples data during read operations. Acknowledgement of all operations is required at the end of a transfer.

OPB is designed to alleviate system performance bottlenecks by reducing the capacitive load on the PLB. Peripherals are generally suited for attachment to this bus. OPB provides the following features:

- Fully synchronous protocol
- Dynamic bus sizing to support byte, half-word, and word transfers
- Sequential address protocol
- Bus parking for reduced latency transfers [14]

OPB will support multiple masters and slaves through multiplexing. This is suitable for a less data-intensive OPB and for adding peripherals to custom core logic. An OPB bridge allows PLB masters to gain access to the peripherals. The OPB bridge acts much like a slave device similar to the bridge in the AMBA architecture. However, an OPB bridge performs dynamic bus sizing, allowing for different devices with different data widths to efficiently communicate.

Finally, DCR is a bus for lower performance status and configuration registers. DCR provides a maximum throughput of one read or write transfer every two cycles and is fully synchronous. DCR is relatively slow, utilizes a ring-type data bus, and provides the required connectivity while minimizing silicon usage [14].

CoreConnect is an open standard technology from IBM's Blue Logic design. Designers have several advantages in creating and using macros that are compliant with CoreConnect. The common interfaces of CoreConnect have allowed for easy integration and provided valuable savings when implementing complex designs.

G. SUMMARY

Understanding I2C, AMBA, HyperTransport, Wishbone, and CoreConnect provides the foundation for developing threat models based on the FHM methodology. Based on the threat models, we can then find similarities in flaws and mitigations that will help us answer the original question asked in the introduction.

IV. IC BUS FLAW GENERATION AND CONFIRMATION

A. INTRODUCTION

Penetration testing will help determine whether IC buses can support system policies regarding confidentiality, integrity, and availability. It is a technique used to evaluate the security properties of a platform to determine whether flaws exist through simulating or conjecturing multiple attacks on a system. The result of penetration testing will ultimately lead to identifying high-risk vulnerabilities, determine the strengths and weakness of its security policy, and provide evidence for additional security countermeasures [15].

We will use FHM as a tool to discover, confirm, and mitigate any potential security flaws from the IC bus protocols discussed in this thesis.

B. FLAW HYPOTHESIS METHODOLOGY (FHM)

FHM is a comprehensive and popular method to conduct penetration testing. It is used in this thesis to determine if there are security relevant flaws in the five IC bus protocols discussed in Chapter III that could violate hardware (i.e., platform) security policies.

The approach to FHM is divided into four stages: Flaw Generation, Flaw Confirmation, Flaw Generalization, and Flaw Elimination. Typically, FHM is conducted by a group of people from various backgrounds and specialties who use Flaw Hypothesis Sheets to keep track of all stages and results during penetration testing. During Flaw Generation phase, they will review organization, architectures and polices of systems to conjecture possible flaws. During the Flaw Confirmation phase, they will prioritize the hypothesized flaws starting from the most severe based on predetermined criteria. Then, they start analytically determining which hypothesis could be true or false before moving on to live testing (i.e., using field-programmable gate arrays [FPGAs], protocol analyzers, etc.). Once all the hypotheses have been confirmed to be true, false, or untested, the group will move on to the Flaw Generalization phase. During this phase, the team discusses the commonalities that were found and the impact that each flaw could have on

a system in hopes of discovering any greater flaws that exist. Upon completion of the Flaw Generalization, the team moves on to Flaw Elimination, in which the group attempts to determine how to mitigate the flaws in order to strengthen the security of the system [15].

Unfortunately, in this thesis, we cannot be completely true to the FHM. The primary weakness in our approach is not the lack of a team (as you can always stimulate a discussion with colleagues) but, rather, the lack of a product against which to test. Thus, we cannot discover implementation flaws (relative to the design), although we can detect design flaws from analysis of documentation. We will concentrate on design flaws discovered through hypothesis of malicious behavior by the bus masters and slaves who have been corrupted. Vulnerabilities due to corrupted bus logic are too broad to discuss in any complete way. This thesis intends to remain as true to the FHM as possible. In the first stage, we will generate suspected flaws that could violate a systems security policy on confidentiality, integrity, and availability from each IC bus protocol. When developing flaws, we must first consider our threat model and a use case scenario for each bus, while keeping in mind the bus policies, specifications, and operations discussed in Chapter III. For every flaw generated, we have to find how or if we can obtain unauthorized services, cause damages (i.e., weaken the integrity), or implement a Denial of Service (DoS) to the user. By answering some questions below that were developed from page 279 of Clark Weissman's paper on penetration testing will help lead us into generating a list of suspected flaws for each of the bus protocols [15]:

- Past experience with flaws in other similar systems
- Ambiguous, unclear architecture and design
- Circumvention/bypass of "omniscient" security controls
- Incomplete design of interfaces and implicit sharing
- Deviations from the protection policy and model
- Deviations from the initial conditions and assumptions
- System anomalies and special precautions
- Operational practices, prohibitions, and spoofs
- Development environment, practices, and prohibitions

- Implementation errors

During Flaw Confirmation, there will be no testing of any of these flaws on FPGAs or equivalent software programs (e.g., Xilinx); however, we will make our assessment on careful analytical research and reasoning that is grounded in supporting documentations (specification manuals, prior research, etc.). We can leave live testing for future thesis work. Upon completion of all flaws confirmed, we will organize them based on priority. Flaw Generalization is where we assess the underlying security weakness found in each bus. This work will be addressed in Chapter V, where we will analyze and compare flaws found and determine if there are any similar or different weaknesses found on the IC buses. Flaw Elimination will be addressed in Chapter VI, where we will recommend analytical and theoretical methods to repair or mitigate IC bus flaws.

C. DEFINING THE THREAT MODEL AND ATTACKER

A design-centric threat model will be used during the flaw generation phase. The attacker sets his/her goals to disrupt the IC bus security policy for which it is set to provide safe and reliable communications among devices through the use of designated reference monitors. Reference monitors are designed to provide protection against unauthorized use and unauthorized modification during bus operations. Examples include protection against service denial, separation of processes, and unauthorized reading or writing to various slaves for which a master is not intended to read and write from. Furthermore, if we assume that if an attack was conducted through subversion, then we can say the entire bus security policy is completely violated. Thus, we will not discuss or mention any type of bus subversion within our flaw generation process.

We must first define our attacker. Simply put, the attacker is anyone! The attacker could be an inside or outside threat, who's an amateur enthusiast motivated to prove him or herself with no money and little resources, or it could be the state- or organization-sponsored hacker with unlimited resources such as logging tools and protocol analyzers. The attacker is looking to break the bus protocol through the discovery of vulnerabilities from its policy, specification, and operation.

We must mention specifically that the attacker will not physically exploit the board circuitry or physical bus. However, we assume the attacker will use user-level processes (i.e., application programs) to attack the bus. Additionally, the threat of a corrupted OS will not be considered as part of our threat model and/or scenarios.

D. I2C FLAW GENERATION

I2C is a free to everyone without any licensing agreements and can be obtained from trusted sources like Philips semiconductors or untrusted sources such as OpenCores.com, Google searches, blog forums, Wikipedia, etc., from which a user of the bus protocol cannot verify how or if the bus protocol was properly tested and verified. They must assume everything works according to specification if they do not have their own verification tools (e.g., protocol analyzer) or a working knowledge of the protocol.

For those designers and manufacturers considered to be a reliable source of the I2C bus protocol, such as Philips, I2C have noted corrupted operation and specification flaws. An attacker can use this to exploit these vulnerabilities to their advantage. On Philips' I2C official website, a section is dedicated to listing all the verified flaws with the I2C buses [16] policy, specification, and operation. Notably, there are a significant number of issues with implementation errors and developmental practices. Implementation errors include unstable power supply and incorrect voltage thresholds that are unable to recognize high and low levels on the SCL and SDA states. Consider a flaw called *unstable power* that will result in DoS, which is referenced from the I2C homepage. A user has a hardware device that carries I2C as its bus and it is unable to run any programs because the I2C bus cannot support the power requirements. That would effectively deny the user any service from the device.

Another significant implementation issue that has been noted by the I2C homepage [16] is that I2C can be too simple of a design if it is implemented on a complex piece of hardware. Imagine a flaw called *oversimplifying* that threatens the bus availability, which we are able to reference from the I2C homepage. Essentially, if a user implemented an I2C bus that is too simple on a complex piece of hardware, then I2C would be unable to effectively support the entire system and its policies because the bus

would be unable to recognize error conditions, arbitration signals, or negative acknowledgements from slave devices. This essentially renders the device useless and leaves the user unable to use the device resulting in DoS.

Additionally, when deviations from the policy and model were discovered during a master device transfer operation, the slave device suddenly stopped acknowledging bytes from the master (which can happen during any stage of the transfer). We can hypothesize an *ignore* flaw that threatens the buses' availability, which is referenced out of the I2C homepage. The user implements a poorly written I2C bus protocol and, during an operation, the slave stops acknowledging data bytes. This would be attributed to a slave device being unable to interpret or misinterpret data from the master device or slave missing a clock cycle [16]. The bus being unable to interpret data results in having to restart the operation from the beginning.

From a security standpoint, we need to be concerned that there is no single manufacturer of the I2C bus protocol. The manufacturing of the I2C bus is inconsistent and there is not one single source to point to or hold accountable if the bus is corrupted, whether it is intentional or unintentional. I2C is a free-for-all and a user must use it at his or her own risk, with no liability and accountability held to the vendor or to the individual that posted the bus code. Each manufacturer and designer of the protocol implements and verifies the bus protocol in his or her own way, which is not consistent with any other designer. The I2C bus developed by Phillips will have a more rigorous process for development, verification, and distribution than that of an enthusiast who wrote an I2C bus code and posts it on OpenCores. From a security perspective, this is not comforting because there is no trust in the product that is being used. This is equivalent to having kids eat unknown candy from strangers. Where is the parent (i.e., the appropriate vendor) to verify that everything is okay?

Inconsistencies of I2C protocol methods of attainment and verification found on the I2C bus ultimately lead us to not trust its policy. The bus protocol can easily be copied from Wikipedia or an unknown developer who posted the code on a blog forum without putting any effort into using a protocol analyzer before implementation. Most importantly, the manufacturers of I2C might not make significant changes. I2C was

marketed as a simple, easy-to-use, and flexible protocol. Significant changes that need to be implemented could destroy its marketability and appeal. Everything read from the I2C specifications manual and official homepage [16] has lead me to believe that I2C has zero security-relevant properties. It is one of the first implementations of an IC bus protocol when hardware security ideas were unimaginable in the 1980s. The specifications, policy issues, and implementation errors cited from its homepage could lead to DoS, thus affecting availability properties. However, without consistency in development, verification, and distribution, this can open the doors to numerous attacks. We can assert that the I2C bus is designed to be told how to act and behave on the IC platform based on preprogrammed protocols, and security and protection is left to the master and slave devices no matter how malicious they can be.

For example, you will not find an I2C on supercomputers, or high-tech equipment. I2Cs are low power and physically very short in length. I2Cs are found mostly in TVs and radios. However, it is plausible to have an I2C on a smart phone or PDA [17]. In our scenario, we have a malicious I2C bus protocol that was taken from an untrusted online source because the manufacturers of a PDA were looking for the cheapest possible build to maximize profits. The I2C bus would have two masters, which are the library and memory, with four slaves, which are the display screen, sensor with digital IO, and two power sources. The I2C protocol was copied from a blog for which the attacker posted it online and had added comments from fake users. The comments posted were boasting how well-written and well-functioning the bus protocol was; however, the attacker failed to mention that the bus was corrupted to allow for the slave devices (screen and sensor) to be exploited.

One of the biggest flaws, and not necessarily an intentional one, is with the protocol policy. We can hypothesize a scenario known as *infinite wait state* that threatens the I2C bus availability. In the I2C specification manual, Section 6.1 (found on page 7) [10], it states that the WAIT state in I2C is designed for a slave (display) to hold down the SCL line and stop all transmission of the master (i.e., Memory) so it may perform some other function or take the necessary time to process data. Only the display device (i.e., the slave that imposed the WAIT state) may lift the SCL and end the wait state;

however, there is no a time limit or function to request a release, or even a reset to the bus. The display can hold the SCL forever and stop all bus transactions on the I2C bus, thus rendering the entire PDA useless imposing a DoS. Figures 20-22 show a pictorial representation that the author designed.

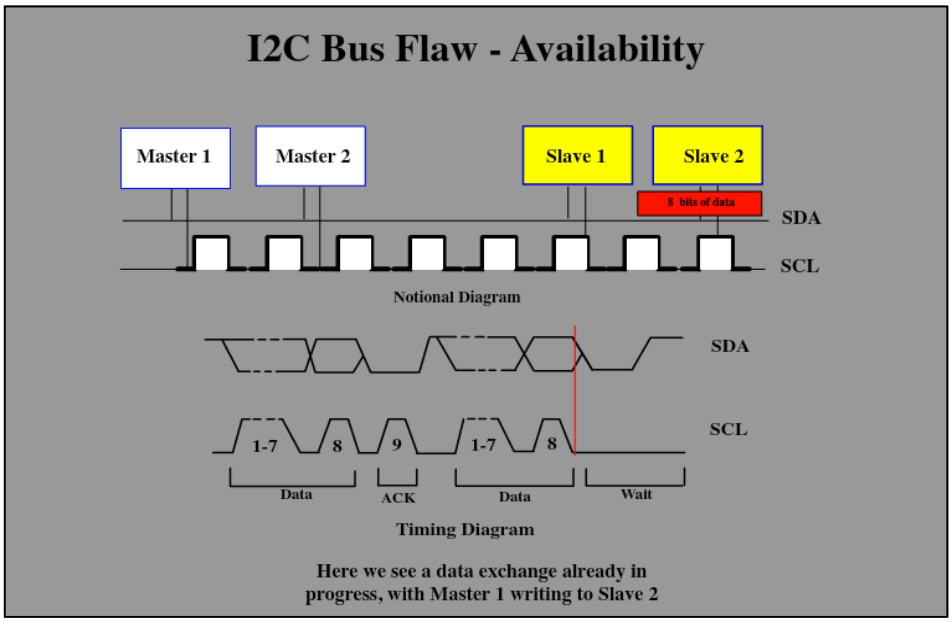

Figure 20. Exchange of data.

During this scenario, a bus operation begins as any other normal operation. Since that was already shown in Figures 4-10, we start with the initial exchange of data between Master 1 and Slave 2.

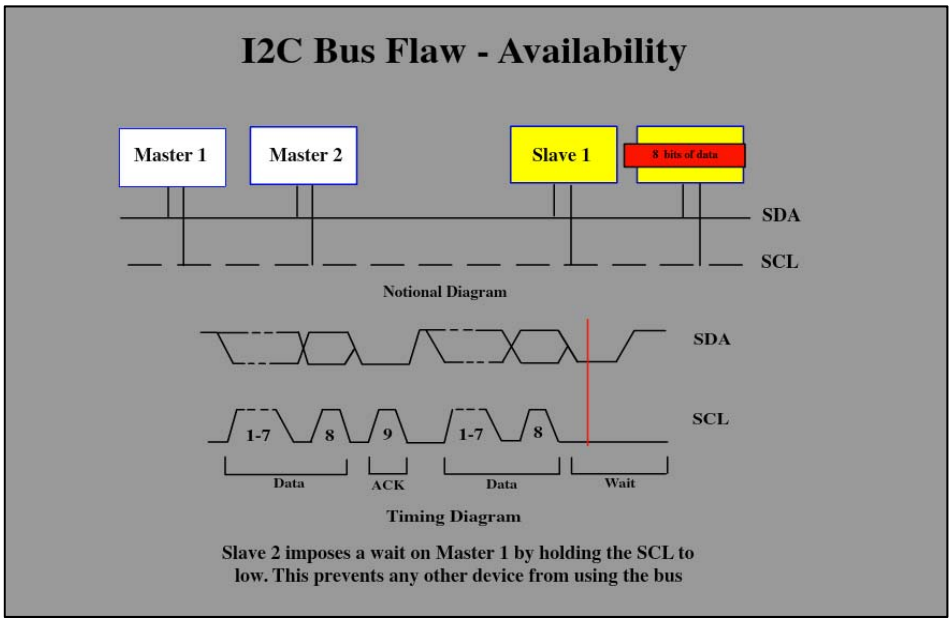

Figure 21. I2C bus in a wait.

As depicted in Figure 21, it is not uncommon to have a bus put into a WAIT state. The dash along the SCL line indicates that Slave 2 has held down the clock line to stop any additional transmission of data.

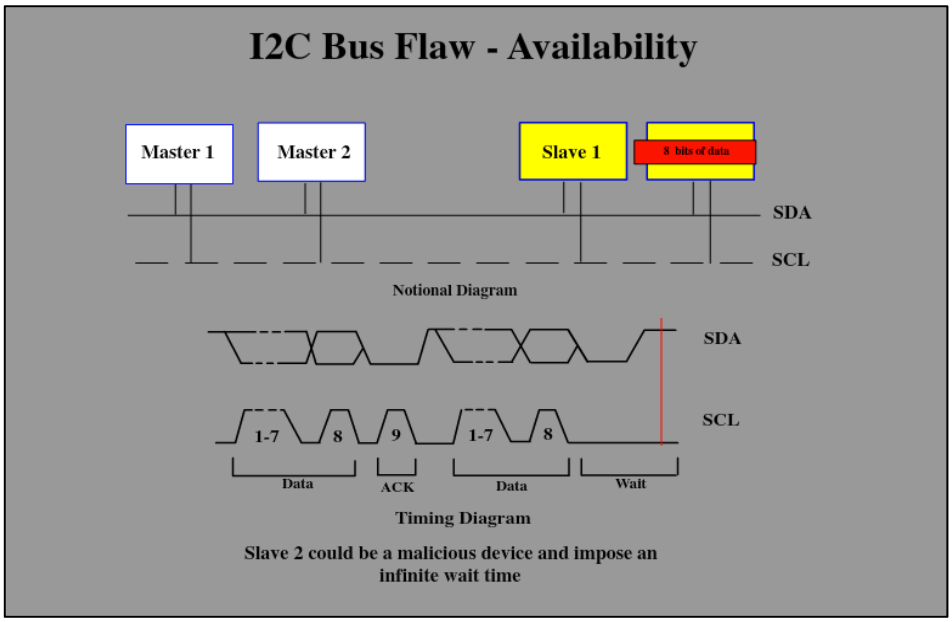

Figure 22. I2C bus in an extended wait state.

In Figure 22, the bus was never designed to limit how long a wait state has to be or can be. This can eventually lead to an infinite amount of time, or the user becomes so impatient that he or she turns off the device to reset the operation, with the risk that it could happen again.

We can generate a *covert delivery* flaw that threatens I2C confidentiality/privacy policy. The design and concept of I2C found in Sections 2 and 3 (page 4) of the I2C specification manual states that the devices share the same wire and respond based on a unique address when information is carried across the SDA line. It also states that the protocol must have some form of communication established in order to not cause confusion among devices [10]. This leads the author to believe that since the devices are on a shared bus architecture and connected to SDA, they are able to see what traffic is being passed and are only responding to what is addressed to that particular device. We can hypothesize a flaw that if there was a misbehaving device on the bus, then this device could perform unauthorized reading of data from a device such as an I/O sensor. If main memory conducts a write transaction to the display, there is nothing that protects the data from being read. Devices are programmed to detect whether the bus is free or busy, in order to prevent a number of collisions. Furthermore, once the transaction has stopped, the sensor has the ability to output data across a network. Specifically, it can be configured to send data directly to a particular network where the attacker is attempting to gain valuable information. Figures 23-26 depict another design by the author demonstrating this.

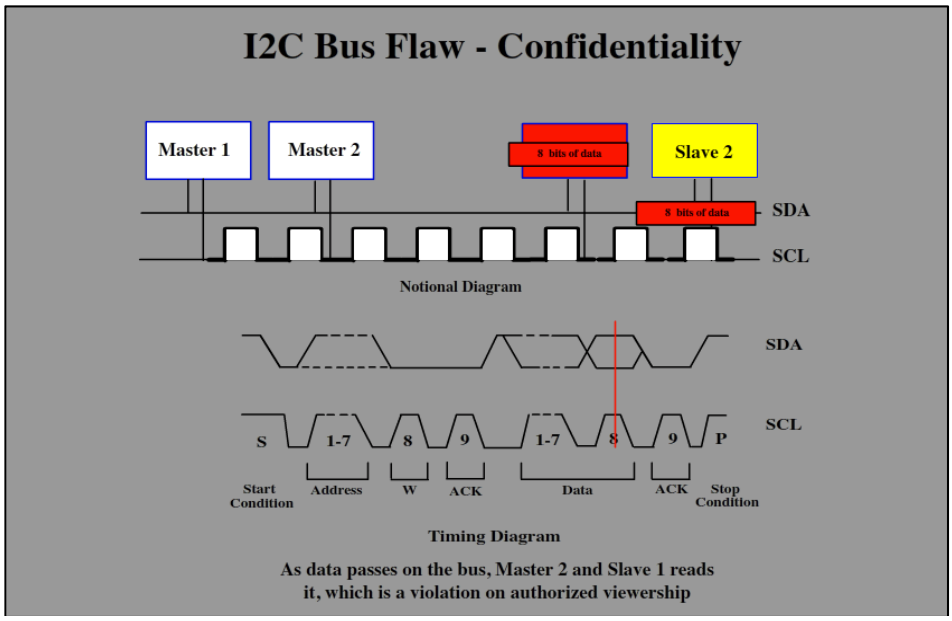

Figure 23. Reading of data.

The operation begins as any other operation. Figure 23 demonstrates that all devices share the same data and clock line. The devices have to be able to read the data line to know whether the bus is busy or free to use.

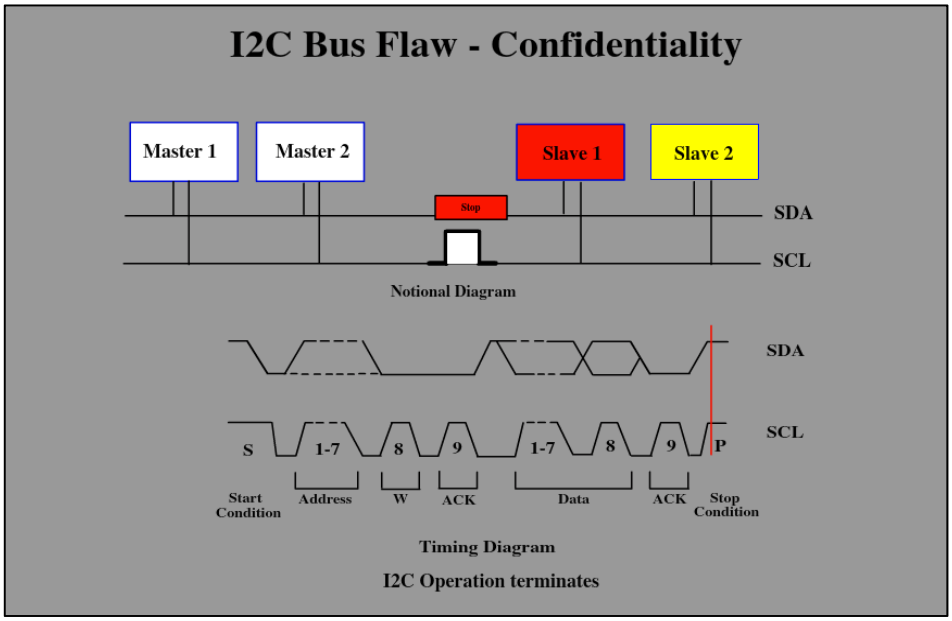

Figure 24. Terminating the bus operation.

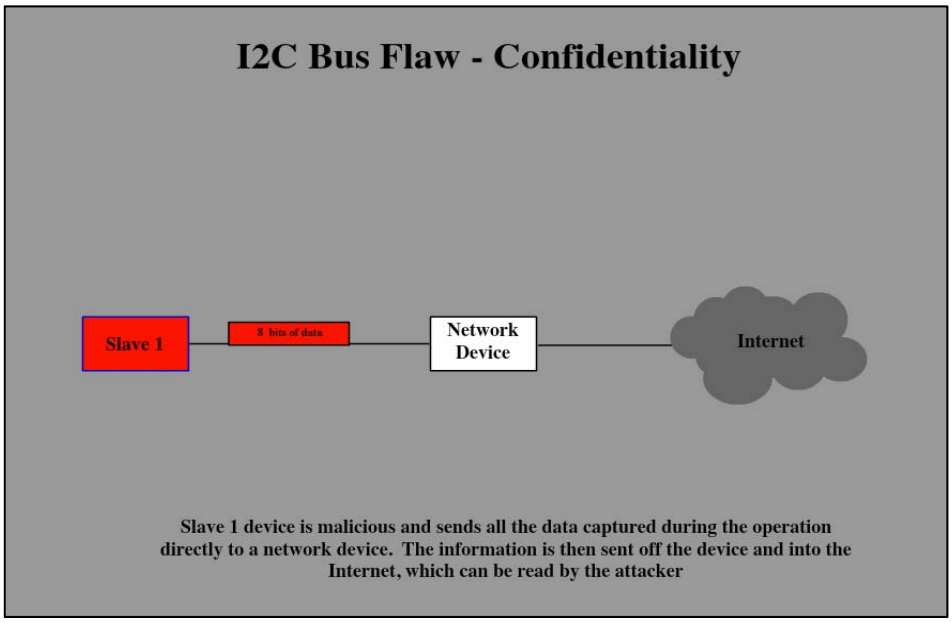

Figure 25. Data sent to a network device.

In Figure 25, once the operation was terminated, Slave 1 sends all the data collected to the network device.

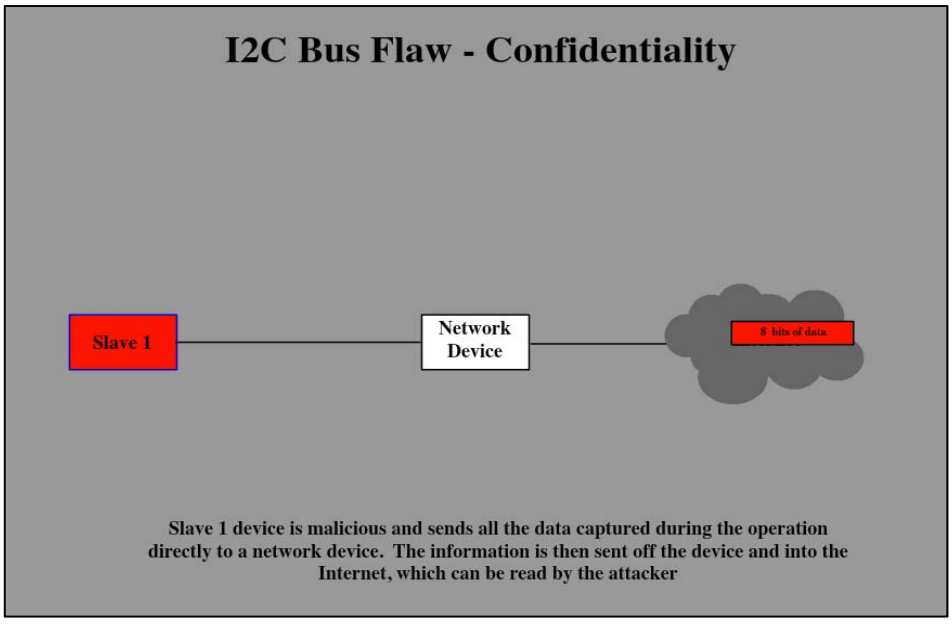

Figure 26. Data being sent to the Internet.

In Figure 26, once the data was sent to the network device, it is automatically forwarded off the network and into the Internet, where the attacker is able read the captured data.

We can generate an *I2C man in the middle attack (MITM),* which threatens I2C authentication and integrity properties. Citing again from Sections 2 and 3 of the I2C specification manual (page 4), each device has to be assigned a unique address. It also states that only one master and one slave can participate on the bus at a single time. However, the design and addressing scheme is left to the designer and manufacturer [10]. In this scenario, the library device could pretend to be the main display as it accepts and sends acknowledgement to the main memory in the beginning of an operation. The main memory will send write data to the library device believing it is the display device. In another scenario, the main memory is writing data to the display (as with the past two scenarios), but the library module is configured to be malicious and can read and have the same permissions as the display device during an operation. The library device is coded to recognize certain bit strings and if a match occurs, it interrupts the operation through initiating a WAIT state on the clock, modifies the data, ends the WAIT state on the clock, and then allows the modified data to reach the display device. Figures 27–31 depict the malicious insertion of data by another slave device.

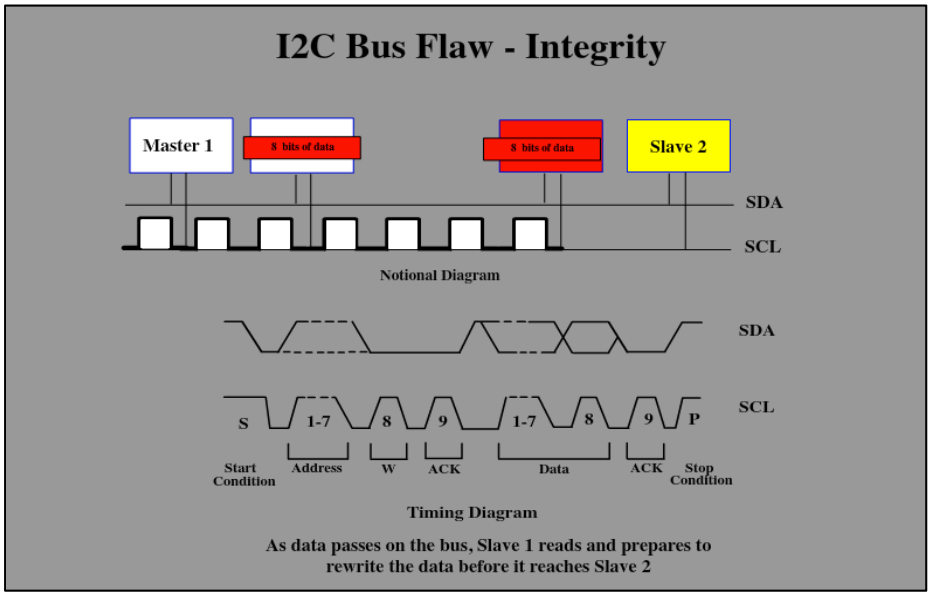

Figure 27. Slave 1 captures data.

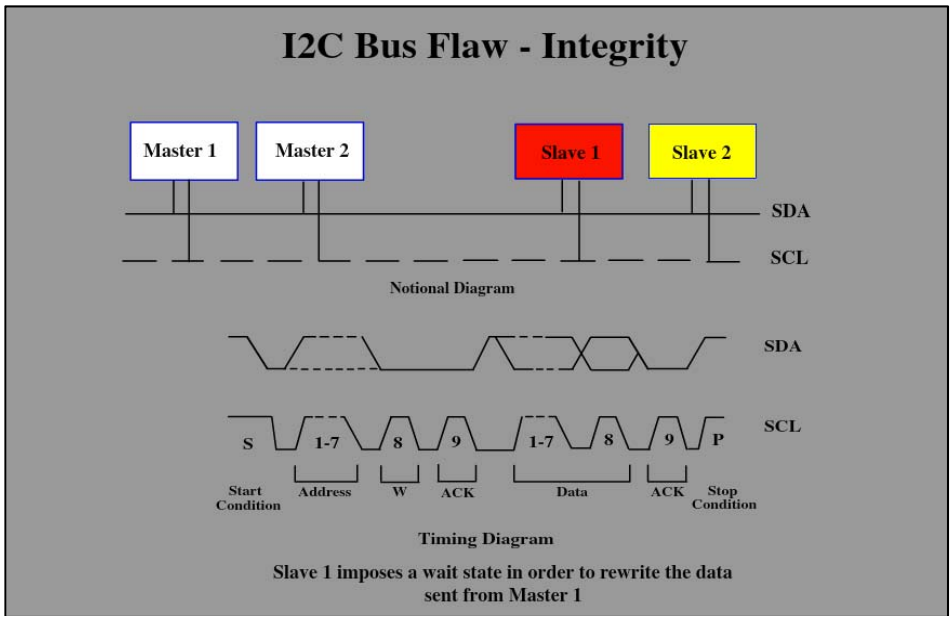

Figure 28. Slave 1 puts the bus in a wait state.

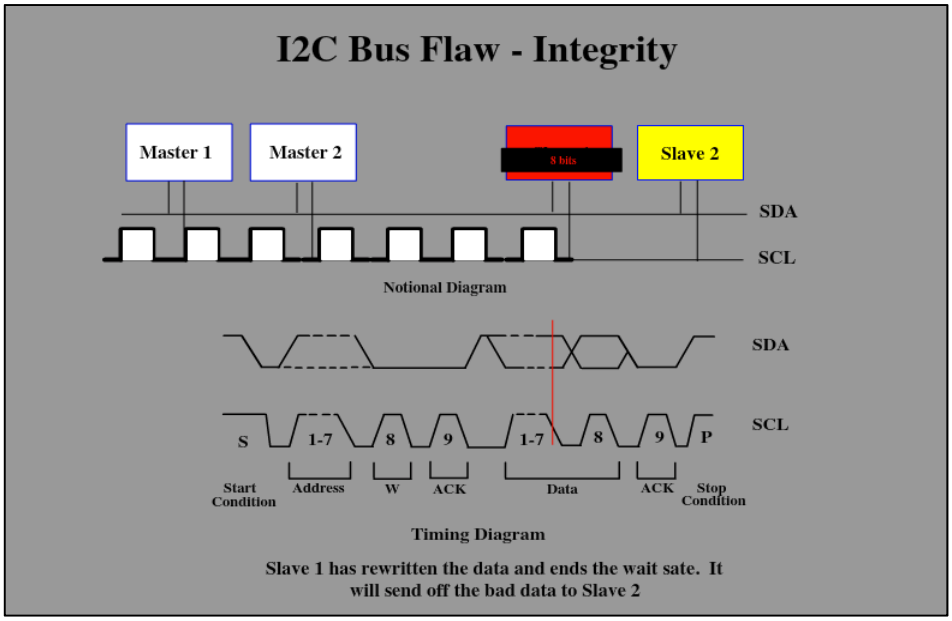

Figure 29. Slave 1 rewrites the data.

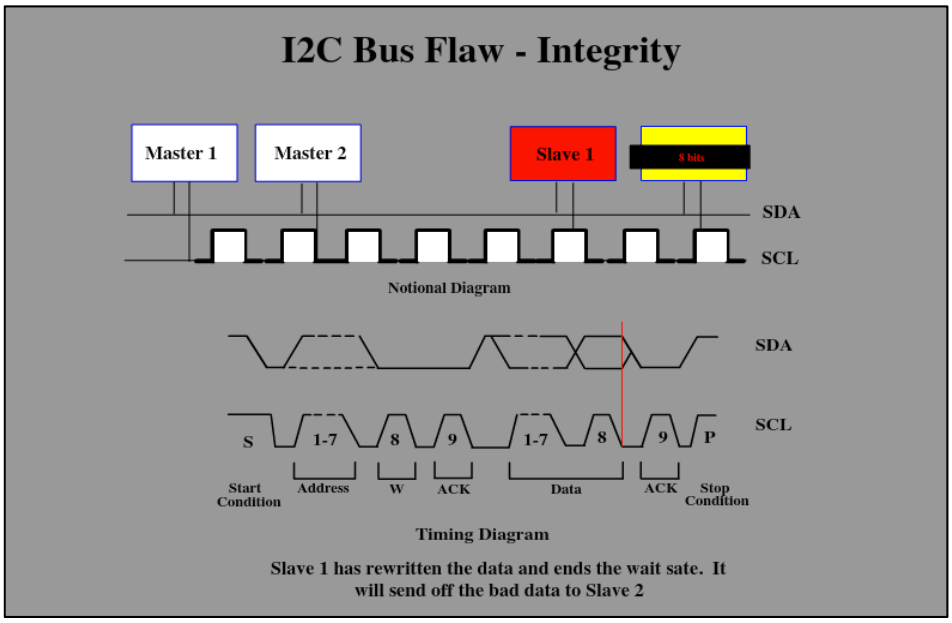

Figure 30. Data is forwarded on to Slave 2.

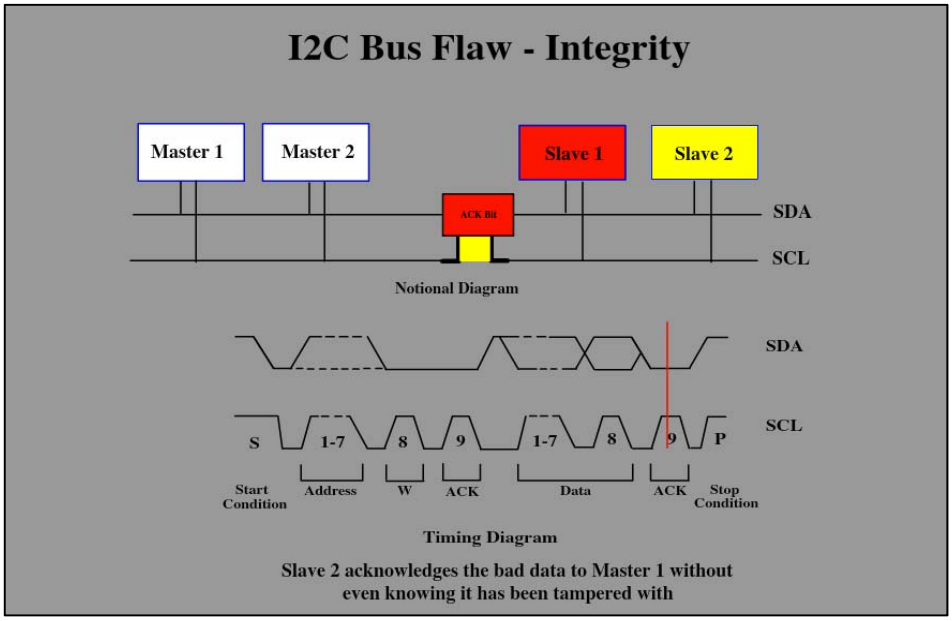

Figure 31. Slave 2 acknowledges the bad data.

E. I2C FLAW CONFIRMATION

We can only partially confirm our hypotheses. While they are untested, we assert they are true based on grounded documentation and logic. Unfortunately, we can never fully confirm these hypotheses to be true or false because implementation testing is not within the scope of this thesis.

The *unstable power* flaw came directly from the I2C official homepage [16]. If the power is unstable, and it loses power, the device completely shuts down. Also, when a device does not directly acknowledge a transaction, then that is, by definition, DoS to the user.

The *oversimplifying* flaw came directly from the homepage of I2C as well [16]. It stated that when buses are configured to be too simple or generic to a complex piece of hardware, then the bus will not respond to commands because it is not programmed to recognize them.

The *ignore* flaw can be confirmed from the I2C homepage [16] because it directly states that during operations, the slave device has been known not to acknowledge data sent from the master device.

Concerning the our *infinite wait state* flaw, Section 6 (page 7) of the I2C manual directly stated [10] that a slave can hold the entire operation in a wait status until it is ready to proceed. Nowhere within the specifications does it state that there is a limit or a reset, which drives our point that this type of hypothesis is theoretically true.

We can analytically assert the *covert delivery* flaw to be true because the design specification of the I2C manual (page 4) [10] only states that the devices need to have assigned addresses to which the information is being sent to avoid confusion. Since I2C is a shared bus, the devices have to see what is on the SDA in order to know if they are being addressed. The author is lead to believe that this hypothesis is theoretically true in that the recognition of devices and unique addressing is left to the designer to figure out. There is no specific guidance. An attacker can design and code this malicious flaw and then publish it.

Finally, given the *I2C MITM attack* flaw, it can be asserted theoretically true based on loose policy guidance found in Sections 2 and 3 of the I2C specification manual [10]. A lot of the protocol is left to the designer to set the communication protocols, which leads to an infinite number of possibilities if an attacker was to design the I2C bus protocol and post it on websites and forums for unsuspecting users to download.

F. AMBA FLAW GENERATION

ARM Ltd.'s AMBA is a designer's SoC bus of choice. They are its only distributers and utilized on most SoCs, and frequently applied as part of research and hardware security testing when SoC devices are needed. The advantage of AMBA as one of the first SoC buses is its protocol familiarity and its product's life cycle is highly controlled throughout development and distribution. ARM also sells a protocol analyzer and other developer tools to constantly monitor and review abnormal protocol activity. ARM Ltd. is constantly improving the AMBA products via its version updates and evolving bus protocols (e.g., AHB, ASB, Advanced eXtensible Interface [AXI], etc.). AMBA is built for reuse and designed to be modularized, which allow for updates as needed with minimal effort for effective integration. This effectively creates Trusted Computing Base (TCB) subsets in order to have and maintain an effective and trustworthy policy.

AMBA holds the advantage over SoC buses discussed in Chapter III because it is not too simple and unverified during its development cycle like Wishbone. Wishbone allows too much independence and trust in developers and users, which can lead to malicious activity. Also, it is not overly complicated like CoreConnect, where there could be some unknown and unintentional policy errors and/or possible leads to covert channels.

AMBA's protocols and timing provides specifications for timely and reliable service [11]. It holds the advantage over I2C in that is has implemented bus resets to eliminate design availability flaws such as infinite WAIT states, interrupts, deadlocks, etc. However, hardware security experts have conducted some extensive research and found AMBA's vulnerability lies within the arbiter and address decoders. Bus arbiters

have fixed protocols, but priorities and accesses are set to the desired policy of the owner of the hardware device. Depending on how the owner configures device priorities and accesses, then the attacker can seize the opportunity to inject a Trojan Horse that can cause MITM attacks, DoS, or unauthorized viewership [18], [19].

In our flaw hypothesis scenario, let us assume that an AMBA resides on a SoC that has been placed on a tablet product. AMBA has the following attached components: DMA bus master, high-bandwidth External Memory Interface, high-performance ARM processor, and high-bandwidth on-chip RAM, which are bridged to a timer, keypad, and peripheral input and output device. AMBA has been configured with a malicious arbitration scheme as an oversight (tools for protocol analysis were avoided to save money). The user attempts to read and write from a text file, which initiates bus traffic.

We can conjecture a *bus hogging* flaw that violates AMBA availability policy given that the AMBA rev 2.0 specification manual states (pages 1-9 and 4-20) [5], "the arbitration protocol is defined, but the prioritization is flexible and left to the application" In addition, (page 3-39) [5] it states that the arbiter can adjust the arbitration priorities and grant signal changes. The *bus hogging* flaw can be exploited by the user intending for the keypad to write to direct memory access via the ARM processor. However, another device is currently using the bus and has been given higher priority over all other devices and effectively continues to hog the bus, to never allow sharing with other devices. This creates a DoS to the user of the bus.

In another scenario, we can hypothesize a *DoS* flaw that affects AMBA's availability policy given the same reference as before. If the keypad accesses the DMA bus master, however, the address decoder is programmed by the attacker to deny any interaction between the DMA and keypad, thus imposing another DoS.

Additionally, we can consider a *full access* flaw that violates AMBA's confidentiality, integrity, and availability policy given the initial references provided if the on-chip RAM is given complete read and write to the DMA device. Configuration policy states that this is not supposed to happen, but if the arbitration policy was

maliciously configured to do so, then this could occur. During bus operations from RAM, the DMA can view, modify, or block. This would violate confidentiality, availability, and integrity of the system.

G. AMBA FLAW CONFIRMATION

For AMBA, we were able to generate three flaws that could be exploited through violating AMBA's arbiter and address decoder. The first flaw was a *bus hogging* flaw that exploited AMBA's policy of allowing one device to have top priority over all other devices, which enables the device to continue to use the bus and not allow other devices to use it—thus effectively creating an availability concern.

With our *DoS* flaw, we were able to conjecture that the keypad device permissions are set to deny any interaction between itself and the DMA. This is because an attacker can allow for the arbiter and address decoder to deny access to the device even though the protocol would stipulate that it should have access to it.

Finally, *full access* flaw allows a DMA device to have complete read and write access to RAM; however, platform policy stipulates it is not supposed to have either read or write access. Again, this comes from allowing the attacker to insert and manipulate decoder and arbitration actions that would allow a device to have full privileges on the bus.

We can assert these entire AMBA flaw hypotheses to be true based on research conducted by L. Kim and J. Villasenor on System-on-Chip Bus Architectures for Thwarting Integrated Circuit Trojan Horses and Trojan Resistant On-Chip Architectures [18], [19]. The research and testing they conducted found methods that would identify run time attacks of Trojan Horses based on unsecure arbiters and address decoders. Specifically, their research proved that through manipulation of arbiters and address decoders, it was easy to manipulate bus and devices access to the attacker's advantage. Due to their research, we can theoretically assert our hypothesis to be true.

H. HYPERTRANSPORT FLAW GENERATION

HyperTransport is different from the five other IC buses discussed and analyzed thus far. HyperTransport connects computer processors to one another and sends data at extremely high speeds. HyperTransport consortium and HyperTransport Center of Excellence is dedicated to designing, maintaining, and upgrading this bus protocol [12].

HyperTransport is a relatively new bus protocol (i.e., developed and distributed within the past 10 years) so it is unexplored in its security-relevant properties. However, a graduate student at Massachusetts Institute of Technology Andrew Huang, was able to test the security properties of HyperTransport bus protocol on an Xbox [20]. In his research, he was able to physically extract data from the HyperTransport bus through an eavesdropping technique known as probing the bus. He then extracted all its contents onto an FPGA and then read from it. The challenge, though, was to probe the high-speed bus without destroying the integrity of the data. Ultimately, he proved that, given knowledge of board circuitry, we can physically extract data from a bus.

Reviewing the architecture and protocol of HyperTransport in Chapter II and Chapter 11 (page 258) of the HyperTransport specification manual [21], data packets, much like network packets, travel up and down a tunnel where each processor in turn reviews each data packet to see if the address belongs to them or not (i.e., data moves in a serial direction). If not, the packet of information will continue to travel to its intended processor. HyperTransport has some of the same characteristics as network addresses, and knowing this information, we can now conjecture that processors have to be assigned addresses for identification.

Image HyperTransport bus was placed on an ATI Radeon Xpress 200 graphics card. The card has an Advanced Micro Devices (AMD) processor, and the HyperTransport bus transports data from one SoC device to other SoC devices in a point-to-point architecture. We can imagine multiple MITM attacker scenarios using HyperTransport. An attacker can exploit a *read-write-deny MITM* flaw that violates HyperTransport confidentiality, integrity, and availability policy. The attacker could have inserted a malicious processor that can be configured to have multiple SoC addresses and

have the ability to spoof another address, just as networks can. If a device wants to send a packet of data down the bus, and if the malicious processor is physically placed before the data's intended destination, it could have an address assigned to read and write to the originator (i.e., spoofing the recipient) or it could just hold on to it, denying the rightful processor access to the data.

I. HYPERTRANSPORT FLAW CONFIRMATION

HyperTransport documentation supports that each processor has to be assigned a unique address and there is ambiguity and leeway pending which device the bus is to implement on. Previous attacks on other systems prove that devices can be configured to spoof multiple addresses and behave as if it were the intended device.

We can assert our *read-write-deny MITM* flaw to be true because according to HyperTransport policy, if the malicious processor is placed in the right position, it must be able read the packet address before sending it on [12]. If the addressing scheme were set to overlap another device, then it would be able to intercept the data before all other processors and read, modify, impersonate, or deny passage of data.

J. WISHBONE FLAW GENERATION

From a security standpoint, the Wishbone architecture and policy suffers from ambiguous guidance, and flexible designs and protocols. Manufacturers have to rely on OpenCores to ensure the design will behave according to the rest of the SoC device policy [13]. Wishbone is built either on a loose set of guidelines and rules, and there is little support to ensure its accuracy, enforce its policy, and clarification on its documentation.

Wishbone also lacks a lot of the advanced default support that other SoCs possess, such as split transactions and power management. Wishbone is designed to be primarily used for small to mid-range embedded systems [13]. Wishbone is relatively low on technology, support, and trust. Realistically, most manufacturers will never use Wishbone on their SoC devices. Wishbone is meant for the mid- to low-level hardware enthusiast who pursues this as a hobby.

The design and policy of the bus is based on the system designer's preferences and specifications, with a limited number of rules that are set by the OpenCores community. In accordance with OpenCores, Wishbone specification manual design requirements include revision levels, types of interfaces, signal names defined, port sizes, port granularity, and a properly filled out Wishbone data sheet [13]. With such low-level guidance, there can be numerous flaws and we can even assert and confirm that any flaw found in a SoC bus, such as AMBA and CoreConnect, would be possible in Wishbone because the policy can easily be manipulated to emulate the other two SoC buses; although Wishbone could include mechanisms to prevent these flaws.

An attacker is easily able to design a weak, flawed, and malicious bus policy and post it to the OpenCores community webpage. Aside from Wishbone's ambiguous and weak policies, its architecture can be easily exploited (consider the three common architecture design flaws discussed in Chapter III). We are going to use the same scenario as we did with AMBA to keep SoC bus scenarios consistent. However, in this scenario, the manufacturer of the tablet is struggling in the current economy and to lower their expenses, they are using a Wishbone SoC bus that they found off on OpenCores.com.

We can speculate a *Point-to-Point (P2P) read-write-deny* flaw that violates Wishbone's confidentially, integrity, and availability properties. The manufacturer selects a P2P Wishbone architecture to implement their SoC. The flow of data will go between three IP cores. The center IP core is the high-bandwidth external memory interface. The other processors have to go through the external memory interface (because point-to-point has to go from device to device in a linear manner); however, we hypothesize that the external interface is untrusted. When the P2P bus was designed, the designer did not define addressing for each device. The external device can easily stop data flow based on destination addresses, or it can simply be reprogrammed to modify certain messages that it is programmed to recognize. All of this would contribute to violations of confidentiality, integrity, and availability.

Let us also consider a *shared bus read-deny* flaw that violates Wishbone's confidentiality and availability policies. The manufacturer downloaded an untrusted Wishbone bus and used a shared bus architecture. Under this schema, only one master is

allowed access to the bus at a time. If the DMA bus master was malicious, it could easily put the bus into a wait state and keep it deadlocked if the designers of the bus did not add a reset function to the bus. Also, since the architecture is shared, there is nothing that can stop other devices from reading from one another while data is passing on the bus. These two possible flaws would violate system availability and confidentiality.

Finally, consider a *Crossbar read-deny flaw* that exploits Wishbone's confidentiality policy. Under the Crossbar bus architecture, there can be more than one master device in operation at the same time, but not to the same slave device. Let us also assume the design of arbitration scheme was rushed. The DMA bus master could have priority over every device on the entire SoC, which would create availability issues. Also, with a poorly written protocol, all the devices on the bus could see the data as it passes through the bus, which creates confidentiality issues.

K. WISHBONE FLAW CONFIRMATION

Wishbone is such a generic protocol with most of the interpretation left to the designer (who may get it from OpenCores), we could easily hypothesis more flaws. The less control, trust, and fewer restrictions on a bus policy, the more flawed it could be. We can assert all of these to be true because the Wishbone specification requirements do not state any guidance regarding protection and manipulating of device and arbitration schemes [13]. Wishbone is also found in OpenCores.com, which is a free website to join, publish, and download [22]. Nothing is verified when published or downloaded, which leaves the victim of a malicious bus at the mercy of the person who published it.

Wishbone suffers from a lack of standards, guidelines, support, and technical specifications. Any designer—whether they are experienced or inexperienced—can write and post a Wishbone bus protocol. The protocol may be designed according to specification, but that does not mean it is a secure bus.

We can confirm a *P2P read-write-deny* flaw that allows a designer to create a P2P Wishbone architecture and have the external interface act in a malicious sense of stopping every piece of traffic to either read from it, write to it, or block it because of poor device addressing assignments.

Similarly, we can confirm our *shared bus read-deny* flaw, which allows for a malicious device to read everything on the bus or have maximum priority over all other devices and thus completely hog the bus from other devices.

Lastly, we can confirm our *crossbar read-deny* flaw exploit when a bus was quickly and hastily designed that allows for a device monopolization on the bus, which essentially causes hogging as well as poorly set permissions that allow a device to read the traffic off the bus.

L. CORECONNECT FLAW GENERATION

CoreConnect is an IBM competitive market response to ARM Ltd.'s AMBA 2.0. They both share similar features (refer to Table 1 in Chapter III); however, CoreConnect has a much more complicated build than its SoC competitors. PLB and OPB buses both have their own separate arbitration schema, with the DCR ensuring that the bus is behaving according to its logic and specification. The devices found on the bus are broken into smaller subsets to help ensure stronger system security and reuse. For example, the external peripheral controller device is divided into three sections. There is a section for Synchronous Dynamic Random Access Memory (SDRAM) ROM, external peripheral device, and external bus memory [14].

CoreConnect is built on complicated logic gates. To ensure that flaws do not exist in the architecture, formal verification must be done. William Lee of IBM and Amit Goel of Carnegie Mellon University conducted a formal verification study on the PLB arbiter. Verification was conducted through a three-phase approach of (1) Modeling, (2) Specification, and (3) Verification [23]. Studies found that the PLB arbiter had a number of vulnerabilities and specification flaws including ambiguity, incompleteness, redundancy, and inconsistencies.

A flaw called *re-arbitration denial* was discovered in which slaves would request for re-arbitration (i.e., secondary request), which has to be passed onto the master, who could ignore it. We can easily hypothesize that an arbiter would fail to pass on the request of a master device for a particular slave device, which leads to a DoS.

Another notable design flaw was that of *large latency* flaws for which the bus performs-locking requests during read and write operations. If we can imagine that if one of the buses were busy, the bus-locking request would be held off, during which another master device can obtain access to the slave before the other master is completely finished.

However, it was also discovered that the master that had its request terminated could regain the bus, which introduces the potential of a separate vulnerability known as a *request termination* flaw that can be exploited for reading and writing data while denying other devices access to the slave device. Problems also arose with timeout conditions throughout various stages of the protocol, which resulted in unknown factors and eventually lead to DoS to the user [23].

Traditional methods of building this IC bus architecture is used to create modular reuse and upgrades as well as smaller TCB subsets for stronger security for platform systems than its competitor, which should ensure a secure and more reliable build. CoreConnect is designed and processed to similar business control solutions such as AMBA. However, like AMBA, the arbitration schema is determined by the manufacturer of the device, which can lead to malicious activity. Any kind of flaw that was hypothesized and confirmed to be true in AMBA could be hypothesized and confirmed to be true for CoreConnect.

CoreConnect is a complicated bus to analyze and understand, but it would seem that its main flaw is that of arbitration ambiguities that lead to integrity and confidentiality flaws.

M. CORECONNECT FLAW CONFIRMATION

We have simply confirmed our hypotheses based on the research conducted by William Lee and Amit Goel from their live experiments that they published. Additionally, much like AMBA, the arbitration and address decoding sequence is similar to CoreConnect based on the CoreConnect specification white paper found on page 7. That explication states “Design toolkits are available for each of the on-chip buses. These tool kits contain master, slave, and arbiter models” [14].

Through the research of Lee and Goel, we can confirm our *rearbitration denial* flaw for which slaves request for rearbitration and the master device completely ignores it.

Additionally, the *large latency* flaw for which caused because a locking request is delayed during a master and slave device operation and because the locking is not set, another master device can interrupt.

A *request termination* flaw can be confirmed for a master device that previously had access to the slave currently in use and can regain access to the bus even though another master and slave should have rightful access.

N. FLAW PRIORITIZATION

With well over 16 flaws generated, FHM calls for prioritization of flaws within each product or system. Typically, we should not prioritize the IC buses because we have not worked with any of their implementations directly; only read, reviewed, and analyzed their specification. However, if we had worked and tested actual implementations of each protocol, we would prioritize them starting with (1) AMBA, (2) CoreConnect, (3) HyperTransport, (4) Wishbone, and (5) I2C. This order is motivated because of platform use and product distribution. For example, since SoCs are commonly found on a platform and AMBA is by far the most popular choice, this should be the first priority, whereas I2C would be the last. I2C is last because it is typically implemented on TV and radio, and damage level and criticality would be minimal compared to other buses. There is no right choice in this matter; it is just what we would have decided to prioritize in this thesis if we were able to actually test these buses.

We can and will prioritize each IC bus flaw based on bus policy violations. The highest priority flaws will be that which exploit bus integrity, followed by confidentiality, and then availability. This is because integrity violations are considered to be more damaging than confidentiality. Again, other teams can prioritize in their own way, but in this thesis, we set our flaw priorities on integrity, followed by confidentiality and availability. Tables 2–6 prioritize each bus flaw.

I2C Flaw Prioritization		
Priority	**Flaw**	**Policy Violation**
1	*I2C MITM*	Integrity/Confidentiality/Availability
2	*Covert Delivery*	Confidentiality
3	*Infinite Wait State*	Availability
4	*Ignore*	Availability
5	*Oversimplifying*	Availability
6	*Unstable Power*	Availability

Table 2. I2C Flaw Prioritization.

AMBA Flaw Prioritization		
Priority	**Flaw**	**Policy Violation**
1	*Full Access*	Integrity/Confidentiality/Availability
2	*DoS*	Availability
3	*Bus Hogging*	Availability

Table 3. AMBA Flaw Prioritization.

Any flaw that we were able to generate for AMBA, we could also generate for CoreConnect, Wishbone, and any other SoC bus.

HyperTransport Flaw Prioritization		
Priority	**Flaw**	**Policy Violation**
1	*Read-write-deny MITM*	Integrity/Confidentiality/Availability

Table 4. HyperTransport Flaw Prioritization.

Wishbone Flaw Prioritization		
Priority	**Flaw**	**Policy Violation**
1	*P2P read-write-deny*	Integrity/Confidentiality/Availability
2	*Shared bus read-deny*	Confidentiality/Availability
3	*Crossbar read-deny*	Confidentiality/Availability

Table 5. Wishbone Flaw Prioritization.

CoreConnect Flaw Prioritization		
Priority	**Flaw**	**Policy Violation**
1	*Rearbitration*	Availability
2	*Large Latency*	Availability
3	*Request Termination*	Availability

Table 6. CoreConnect Flaw Prioritization.

O. SUMMARY

In this chapter, we explained FHM and conducted the first half of the process. Chapter V will attempt to generalize flaws individually discovered in hopes of finding bigger flaws and discover overarching issues with the IC buses in an effort to mitigate these vulnerabilities in order to strengthen the security of hardware platforms.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FLAW GENERALIZATION

A. INTRODUCTION

Our analysis thus far leads us to believe that all buses are designed and built to serve one purpose, even though there are different methods on how to do so. Essentially, a bus is a service provider for those devices or processors placed on the platform. Ultimately, one device needs another device to perform a read, write, or execute instruction; otherwise, nothing gets done. IC buses are designed to ensure that data is delivered from source to destination in a quick and reliable manner, which includes ensuring that the confidentiality, integrity, and availability security policy of each IC bus is adhered to.

In this chapter, we generalize the exploitable flaws we have generated and confirmed from Chapter IV in hopes of discovering the impact and greater common vulnerabilities that lie within general-purpose IC buses. Determining the impact and the big picture of a IC bus flaw will not only help us understand our generated flaw better, but perhaps find system-wide approaches that ensure stronger platform security through mitigation of that flaw.

Our approach will be to find and identify the impact and general causes of exploitation with each flaw generated in Chapter IV, which will lead us to identify security vulnerabilities with the goal of flaw elimination in Chapter VI.

B. FLAW GENERALIZATION

We will generalize every flaw sequentially in the order they were generated. Beginning with the *unstable power* flaw on the I2C bus, there was nothing wrong or intentionally malicious with the bus itself. However, if a user's system were to have this flaw, the entire system could cause a completely unexpected and unrecoverable shutdown resulting in loss of all data. Depending on the system and data, the impact could be minor (e.g., a one page memo) or catastrophic (e.g., processing highly sensitive data). The bus could work perfectly well on some platform that requires less power. The user of the I2C protocol has to be aware of the protocol's capabilities and limitations.

The *oversimplifying flaw* is similar because there is nothing physically wrong with the bus. We can assume it was written correctly and that it works to specification. We can stipulate that a user did not choose a robust enough I2C protocol to implement on their platform. However, if a system was to have this flaw, it could ultimately lead to a platform's coming to a standstill. I2C does not have any reset capabilities, which could force the user to perform a hot reboot that could further damage the system. The user might also face a loss of data and, depending on what data the system was working on, the impact can range from minor to severe. The lesson learned is that the user must do a better job of choosing a bus to put onto their platform.

The *ignore* flaw can be attributed to bad programming. We can assume that there is nothing intentionally malicious, as with the previous two flaws. However, if the *ignore* flaw were to take place, the consequences could have results similar to the *oversimplifying* flaw. The attacker was able to exploit the *ignore* flaw because the programming was weak and the programmer took shortcuts that included failure to ensure that components and devices could interact as designed.

The *infinite wait state* flaw is a repercussion of the I2C protocol design. It was one of the first IC buses to enter the market in the 1980s, when platform security and networks did not exist. *Infinite wait state* will have the same impact on a system as the *oversimplifying* or *ignore* flaw. We can infer that during the design process the designers skipped over reasonable security and safety measures that would prevent the bus from suffering from any long-term interrupts. Other bus protocols that offer reset capabilities have come to the market. The designer needs to use stronger methods for testing and analysis to ensure bus reliability.

A *covert delivery* flaw is the consequence of having all devices share a single bus architecture. If a *covert delivery* flaw were to occur, the impact would be severe. Information is sent from a secure system to the attacker's system. The information could be made public (e.g., country A will attempt to assassinate the president of country B) or put to the attacker's advantage (e.g., the army of country A will patrol location X at time Y). The shared architecture design drawback is often discussed in scientific papers, journal entries, and conferences explaining and demonstrating how tests proved that a

shared bus can be capitalized on in order to exploit a system's confidentiality policy. Most designers would like to believe that all devices attached to a shared bus behave as designed.

I2C MITM flaw is possibly the most vicious of all exploitable flaws on I2C. The impact of the *I2C MITM* would have the same consequences as the *covert delivery* flaw, in addition to the attacker gaining the advantage of the ability to modify the stolen information and send it back onto the bus without being noticed (e.g., instead of the message, country A will attempt to assassinate the president of country B, the information could be modified to read as country A will attempt to assassinate the president of country C). This flaw is possible because I2C allows for a shared architecture in which other devices can read what is on the bus. If an attacker were able to access a device's "root" permissions, then they would be able to have complete control of the bus, which allows the malicious device to have more control than the reference monitor. Designers need to ensure that strong arbitration rules and security bus logic are in place to establish that the bus behaves as designed.

The *bus-hogging* flaw is attributed to allowing the manufacturer of the platform the freedom to set the priorities of the devices placed on the bus. If this flaw were to occur, the efficiency of the bus would be severely degraded because only one device is actually getting data processed, while others are unable to access the bus. However, the impact is minimal because AMBA has a reset function. The user can detect the error, apply the reset function, and administer changes as necessary to resolve the system failure. The security concern is allowing trust in a manufacturer or the attacker to have flexibility in the operational implementation of the bus. With just an "out of the box" bus (i.e., no protocol analyzers used), the flaws can be severe.

Our *DoS flaw* is caused by the failure of arbitration or reference monitor schemes being unable to adequately recognize or authenticate particular devices on the bus. The impact of the *DoS* flaw is similar to the *bus-hogging* flaw. This can be considered a symptom of the AMBA authentication and arbitration protocol failing to recognize the devices on the bus and their permitted accesses. The impact is minimal because the reset

function is available, and if a user or manufacturer has bought a protocol analyzer, then testing can be done to determine the error and fix it.

The *full access* flaw is severe. If it were to occur, a device can read secrets from other devices on the bus (i.e., compromise the entire system). It will also deny information processing in which a user will not get the vital information needed to make critical decisions. Furthermore, the *full access* flaw would permit a device to send misleading instructions to another device, which could process corrupted information or cause the entire system to crash. The flexibility in AMBA's protocol gives the attacker the ability to grant device permissions that allow for a malicious device to read and write from a device not specified within the manufacture's design parameters.

Our *read-write-deny MTIM* flaw on the HyperTransport bus can be exploited because of HyperTransport's P2P architecture, and it allows every device or processor to be its own separate reference monitor. If this flaw were to occur, the consequences could be dire. HyperTransport is a high-speed, powerful bus architecture, and if a device were to be exploited, sensitive information could be exposed or malicious code could be processed, resulting in an unrecoverable crash, memory wipe, etc. The result could be that users lose valuable information such as financial data. Unlike a shared bus architecture, not all devices have to see the data on the bus; only the devices that fall between the sender to receiver. Each device is required to quickly process the address space of the data packets to know if it is destined for it or not. Designers have to be careful in specifying address ranges for each device.

The *P2P read-write-deny* flaw is similar to the HyperTransport *read-write-deny MITM* flaw. They both share nearly the same architectural properties and the same generalities. The difference between the bus architectures is that a designer of Wishbone has greater freedom to add security features to the bus. Depending on the implementation, the impact of this flaw could be minor because the bus resides on a standalone device, or it could have the save impact as the *read-write-deny MITM.* The concern is that if a user were to apply the Wishbone bus protocol, there is no technical

manual to consult and troubleshoot errors, and there is no vendor to consult for help. Additionally, this flaw could be the result of the designer's poor programming practices and taking shortcuts.

Our *shared bus read-deny* flaw shares the same generalities as I2C *covert delivery* flaw. The impact of the flaw would be moderate. The users could lose system efficiency and functionality as well as reveal sensitive information to the attacker. I2C and Wishbone have similar architectures and weak reference monitoring. Furthermore, this flaw does not have to be intentional. Bus designs such as these can be victims to amateur programmers who have done a poor job in generating the proper design requirements and/or lack the programming skills to ensure that the protocol meets its intended specification.

The *crossbar read-deny* flaw exploits bus policy by allowing more than one master on the bus at a time. The impact of this flaw would be similar to that of the *shared bus ready-deny* flaw. Using crossbar architecture, it can be programmed and distributed to the user without a robust reference monitor to arbitrate between both master devices when they want to utilize the same slave device. In the case of a lack of protocol analysis, manufacturers can easily miss that a master device could have more permissions than allowed. The *crossbar read-deny* flaw need not be intentional. Since this flaw occurs with the Wishbone architecture, the user can fall victim to poor design and programming techniques. A user has to be aware of what kind of protocol they are using and accept the risk they are taking by using the chosen protocol.

Our *rearbitration denial* flaw stems from IBM's CoreConnect being easily corruptible. The impact of this flaw would likely be low. The user will lose system efficiency, but the user information on the devices is safeguarded. CoreConnect is a very complicated SoC mechanism. With complicated architectures such as CoreConnect, it is difficult to detect flaws and trace the root of a flaw's error. These types of flaws are not intentional, but rather overlooked during the implementation phase of development. *Rearbitration denial* occurs when a small subset of a device does not act according to its specification. Flaws such as these are often hard to detect without thorough analysis.

The impact of a *latency flaw* is similar to a *rearbitration denial* flaw. Although the device may have been designed with the best of intentions, due to an oversight during implementation and testing, a programming error is overlooked or deemed minor by the designers. CoreConnect allows for a reset function, and technical support at IBM is available.

The *request for termination* flaw is a result of the same issues as the other two flaws on the CoreConnect bus. The system would suffer from reduced efficiency, which we consider a low impact. The reset function and vendor support are available.

C. KEY POINTS OF DISCOVERY

Our approach is to generalize the overall security of each bus protocol in an effort to answer the question in Chapter VII of whether or not one bus protocol is more than secure than another.

With the exploits of the I2C bus, we found that the flaws were a result of untrusted sources causing simple design corruption. Other causes include a lack of requirements during the design process, poor implementation techniques, and a user not knowing how to pair the right bus with the right platform. Additionally, trusted sources' exploitable flaws, such as Philips Semiconductor's, primarily result from poor policy requirements.

AMBA flaws are a result of poor design and overreliance on the reference monitor. The designers have put too much trust in allowing manufactures to have flexibility of device preferences. AMBA has to have some degree of flexibility in order to maintain high marketability across several products. However, if there are no protocol analyzers or other tools involved, these small nuances can be easily missed, and an attacker can exploit the reference monitor for personal gain.

Based on an understanding of the *read-write-deny MITM* attack, HyperTransport makes poor use of reference monitoring given the architecture. Other buses seem to have a separate entity, such as an arbiter, that acts like a reference monitor to ensure correct device behavior, where HyperTransport requires that each device monitor itself.

Additionally, bus architectures like HyperTransport will allow individual processors full access to plaintext data packets if addressing assignments are not well defined.

Wishbone bus flaws share similarities with AMBA, HyperTransport, and I2C; however, these flaws are easily avoidable. No two Wishbone buses are the same. There is no exact specification, manual to consult, or technical support team to call on. Wishbone flaws can result from bad design practices, poor programming, users having little knowledge of the bus protocols, or failure to apply methods for analyzing the bus prior to implementation in hardware.

CoreConnect's design complexity makes it difficult to perform thorough security analysis. Complicated circuitry can often contain errors and provide attackers with an opportunity for exploitation. However, we argue that the generalities and key points related to AMBA can also be applied to CoreConnect.

D. SUMMARY

We have examined the flaw generalities and impacts from the 16 exploits that we generated and confirmed in Chapter IV. Overall, we found security issues that pertain to a product's life-cycle development, including insufficient design requirements, lack of testing, failure to determine the most appropriate bus for a given platform, and poor reference monitor design. In Chapter VI, we shall discuss flaw elimination techniques and which techniques can be used to enhance IC bus security.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FLAW ELIMINATION

A. INTRODUCTION

We will analyze and discuss various technologies and techniques that can be applied to IC bus architectures and discuss their benefits and drawbacks. We will first look at what existing techniques have to offer, including golden model, bus encryption, and redesign. We will also suggest some emerging technologies from which IC buses could benefit, including Security Enhanced Communication Architecture (SECA), TrustZone, Parallelized Encryption and Integrity Checking Engine (PE-ICE), and Gate Level Information Flow Tracking (GLIFT).

B. GOLDEN MODEL

Professor Mohammad Tehranipoor of the University of Connecticut published a survey of hardware malicious inclusions [24]. In his paper, he stipulated but did not promote that using a golden model to verify hardware is one method to mitigate malicious inclusions that could violate a system's security policy and compromise confidentiality, integrity, and/or availability of a system.

Using a golden model would consist of taking each bus protocol and verifying every step and every scenario. Once the IC has been verified, it will be forever deemed "The Golden Model" because it would be free from flaws, corruption, subversion, and other vulnerabilities. Once the golden model of the IC bus protocol is in place, every bus manufactured after it has to be compared to it; the basis of comparison includes, but is not limited to, logic gate comparison, functional comparison, power threshold comparison, and clock cycle comparison.

We can consider the golden model methodology as a base case because using it would effectively eliminate every flaw. However, as stated by Professor Tehranipoor, it is extremely inefficient and impractical, and verification is only good as the person and tools that verified it. The time it takes to test and verify a single golden model and then compare it to every IC manufactured would slow down the manufacturing and the distribution process. Consider how long it would take to compare 100,000 IC buses to

one golden model. This would make the manufacturing and distribution process of AMBA, CoreConnect, and HyperTransport impractical. We can consider using a golden model on a bus such as Wishbone or I2C because demand for them is relatively low, and they are both fairly simple to compare and verify. However, because the product does not have a single source vendor like the other IC buses, it would be untenable.

A golden model is a nice concept in theory, but impractical, inefficient, and would never work because of the general nature of manufacturing and distribution. Users would never wait so long for a product and would deem it costly for vendors. The risks in using the golden model far outweigh the rewards. The major problem with the golden model is that it is impossible to know for sure whether the golden model itself is free from flaws.

D. REDESIGN

Redesigning bus protocols to eliminate weakness is another solution, but is much easier said than done. Specifically, Lok-Won Kim and John D. Villasenor [18], [19], suggest multiple redesign methods on AMBA that include reconfiguring the address decoder, reconfiguring the arbiter, adding a bus matrix, and adding input signal wrappers. These methods are also applicable to other SoC bus protocols.

Kim and Villasenor suggest reconfiguring the arbiter using a fixed priority scheme, a round robin scheme, or a combination of both. Using a fixed priority scheme would eliminate the flexibility for adjusting device priority and access. This will eliminate confidentiality and integrity vulnerabilities found on SoC buses and provide an effective and noninvasive technique that results in no loss of system performance.

Using a round-robin scheme, in addition to fixed priority, provides an additional benefit because it sets a specific amount of time that a master can utilize the bus without hogging the entire bus and effectively denying other devices from using it. This would be a popular technique to implement since every device will get a fair allocation of time on the bus. There might be some system degradation because some devices might need to use the bus more often; however, a designer can easily implement and adjust the round robin time share based on frequency of use. Figures 32 and 33 illustrate this round-robin method.

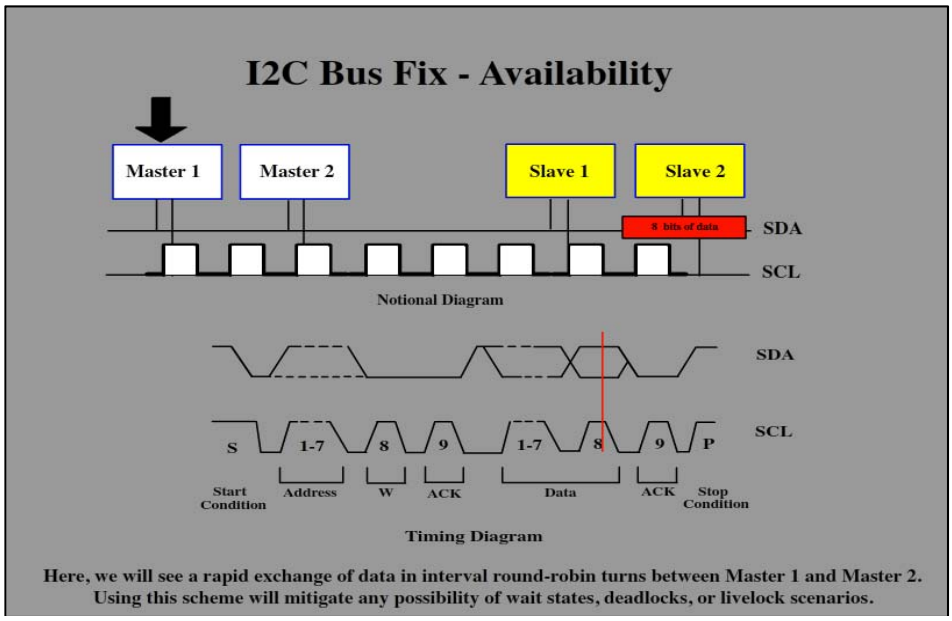

Figure 32. Round Robin Master 1.

In Figure 32, the arrow above Master 1 indicates that it is currently involved in a transaction with Slave 2. Using a round-robin scheme conveys that each device has a set amount of time to use the bus at one time. No matter if the bus is in midoperation, if the Master device's time has expired, it must cease operations and allow Master 2 to start as seen in Figure 33. Once Master 2 is done or time has expired, Master 1 may continue its previous data transmission. Using the round-robin and fixed-priority technique effectively eliminates availability vulnerabilities on any bus architecture.

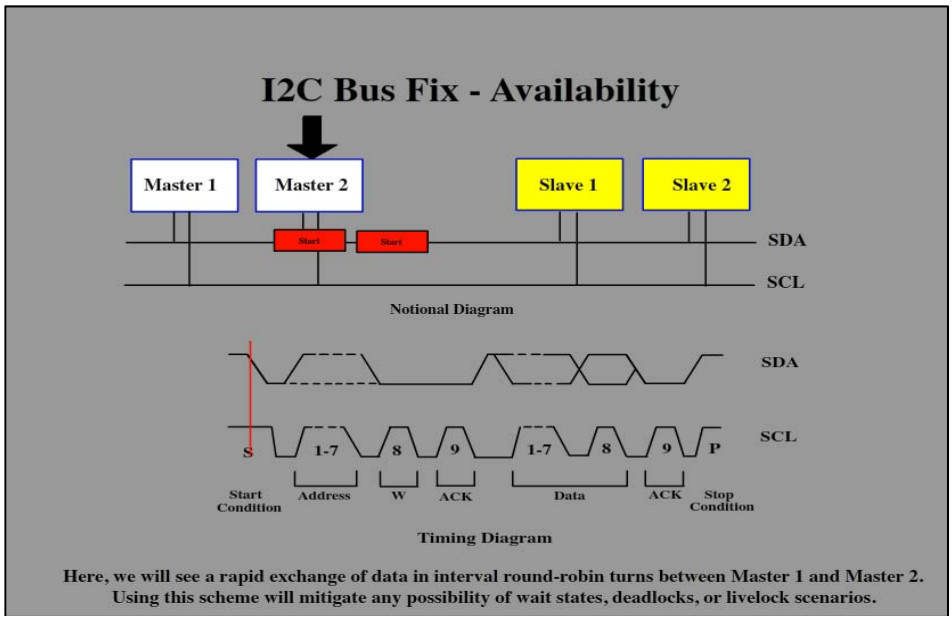

Figure 33. Round Robin Master 2.

By modifying the address decoder, a designer can set the decoder to detect an attempt by a malicious bus master to access a restricted address and be able to block a normal master from inadvertently accessing malicious slaves [18], [19]. This scheme will allow for the detection of a malicious Trojan slave, which will then be disconnected from the bus. If a master attempts to access a malicious slave, the address decoder will divert the access to a default slave containing empty address ranges, thus effectively excluding the malicious slave from the system [18], [19].

Additionally, implementing a secure bus matrix enables a connection between the authorized master and slave device. A secure matrix will detect, block, and report a malicious wait signal from a malicious Trojan device. Using the secure matrix will effectively eliminate any availability vulnerabilities associated with a SoC bus policy.

Reconfiguring system logic to provide input signal wrappers between master and slave devices can prevent eavesdropping and modification by unauthorized devices on a shared bus architecture. The nature of shared bus architectures leaves the entire platform open to vulnerabilities. Logic is provided such that data is only visible to the master and slave legitimately involved in the bus transaction [18], [19].

This technique would have to be used in conjunction with another type of flaw elimination method that would ensure the availability of the bus. Figures 34 and 35 demonstrate input signal wrappers.

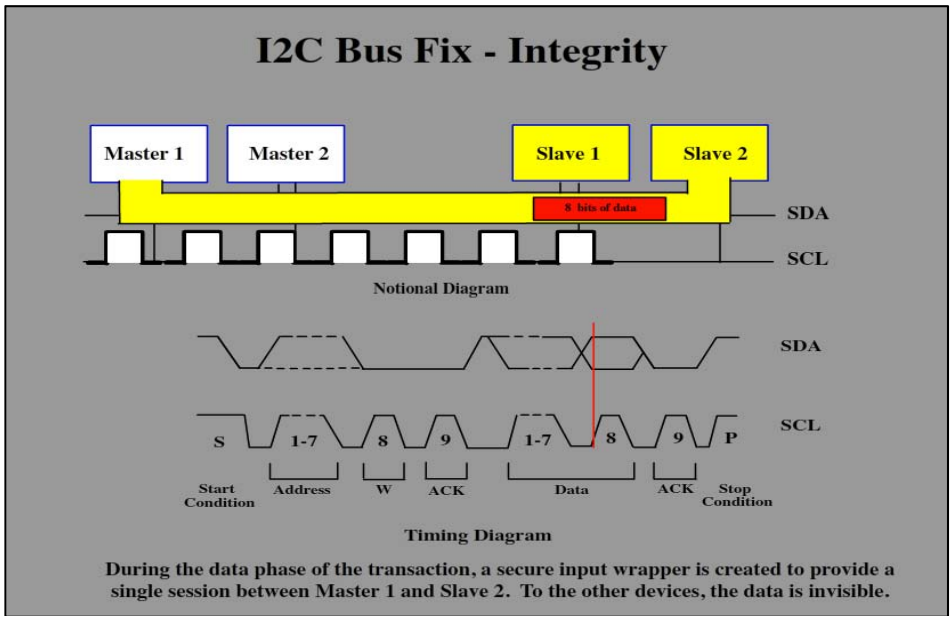

Figure 34. Input Signal Wrapper.

In Figure 34, the big yellow block represents a logic input signal wrapper such that only the data transaction is visible to Master 1 and Slave 2. Once the data transmission is complete, the input signal wrapper is gone and the bus architecture returns to normal, as seen in Figure 35.

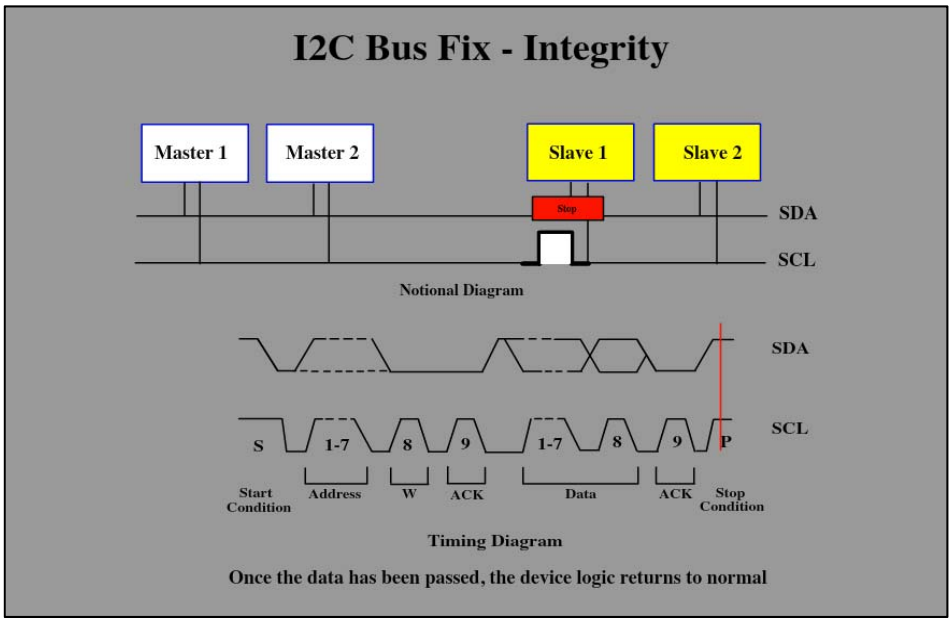

Figure 35. Signal Wrapper Termination.

These policies and architecture changes are effective and rather simple to implement; however, the research and testing for these solutions were conducted on SoC IC buses and not on buses like I2C and HyperTransport. Design options such as input signal wrappers and round-robin schemes are applicable, but a redesign of arbitration methods is not.

E. BUS ENCRYPTION

Encrypting data on the bus is one the simplest and most practical methods for eliminating confidentiality vulnerabilities. Advanced encryption algorithms already exist that have proved to be efficient. For example, a designer can use a Rivest, Shamir, & Aldeman (RSA) encryption algorithm that uses asymmetric key exchange of symmetric keys between devices, with each device already having its own asymmetric keys preinstalled. An operation will begin with the master device sending an address request to a slave device, along with its signature. In turn, the slave device will send an acknowledgment and its signature to the master device. Once the identities of the master and slave device have been confirmed, the master device will send information encrypted with the slave's public key. The slave device will receive the information and decrypt it

with its private key. For more efficiency, the master and slave can use asymmetric cryptography to exchange a symmetric key. Figures 36-39 demonstrate this idea.

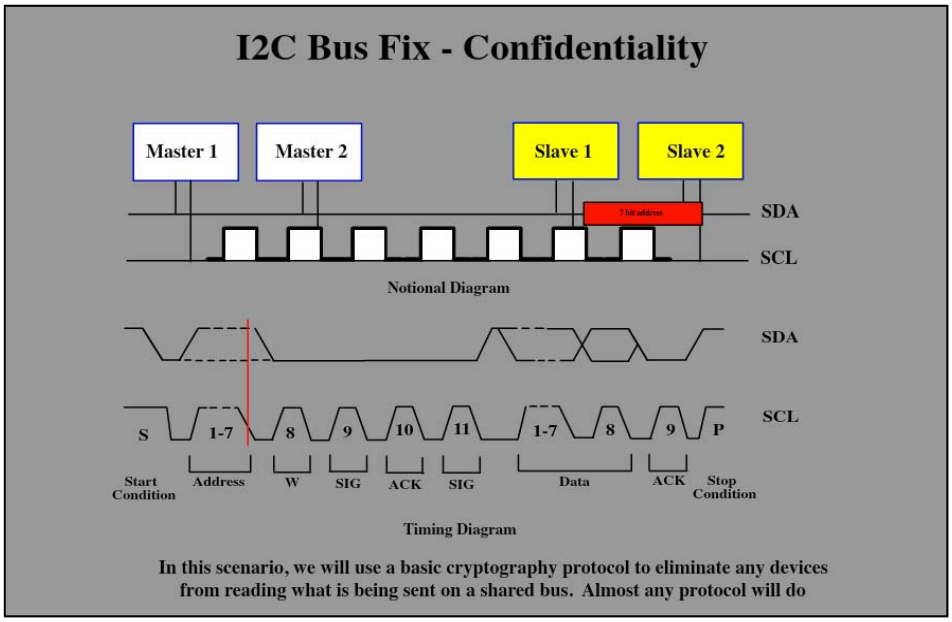

Figure 36. Initial Data Transfer.

In Figure 36, Master 1 is sending a request to Slave 2 for a data transfer. This is how any normal operation would begin for I2C.

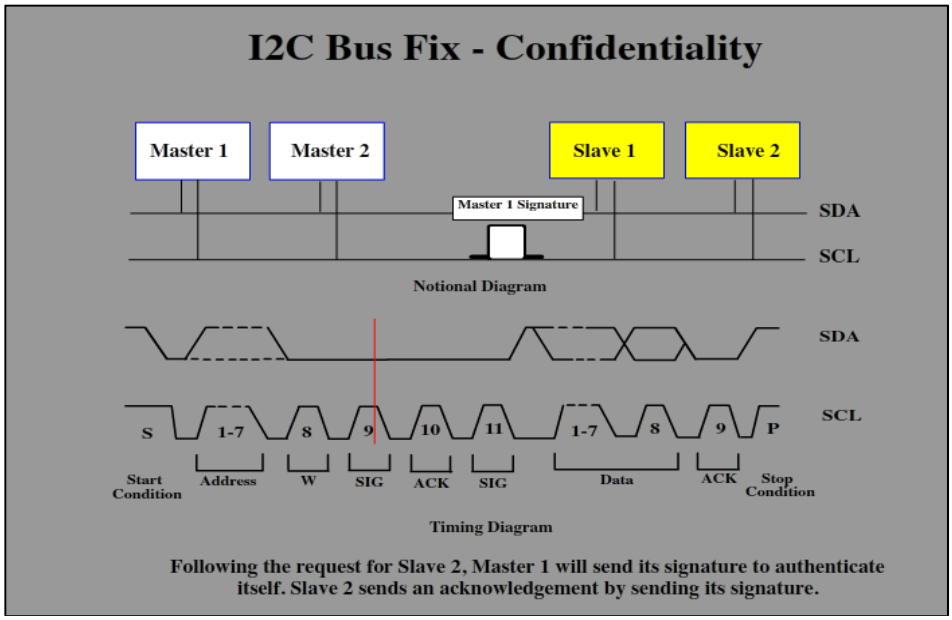

Figure 37. Master 1 Signature.

After Master 1 has sent its request for a data transfer to Slave 2, it sends an additional bit that will indicate the device's signature in order to authenticate itself to Slave 2.

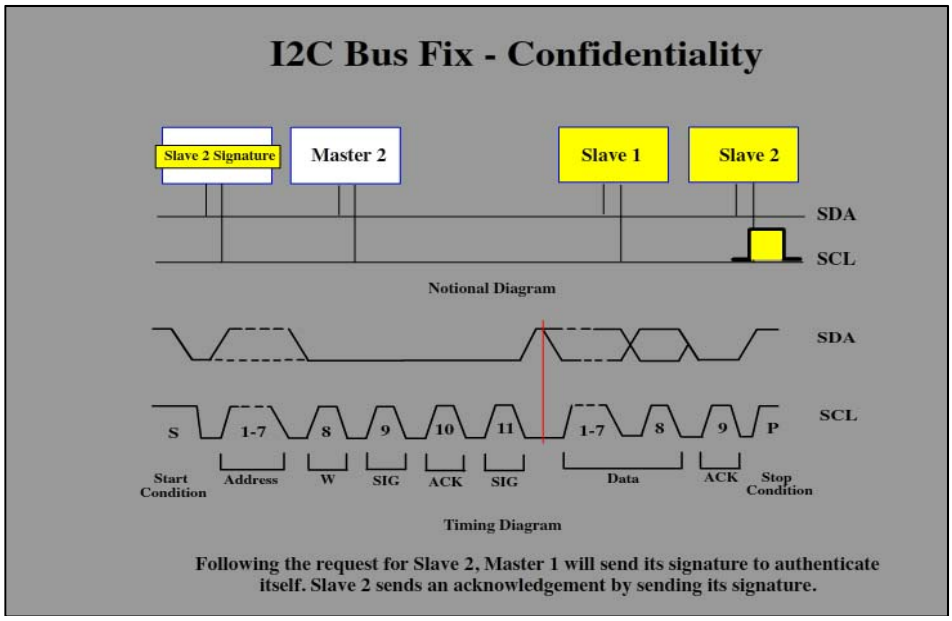

Figure 38. Slave 2 Signature.

After Slave 2 has sent an acknowledgement bit to Master 1, it sends an additional bit to indicate its signature, as seen in Figure 38.

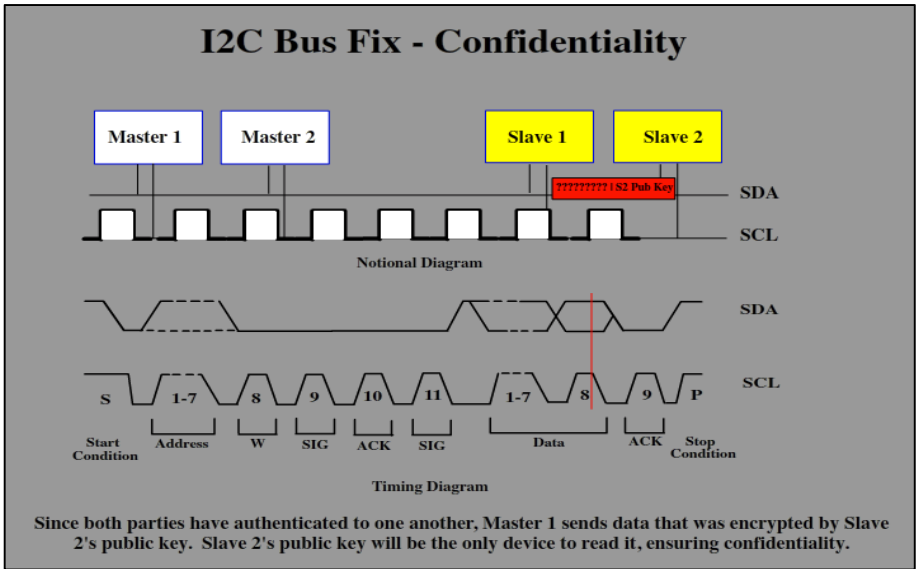

Figure 39. Encrypted Data Packet.

In Figure 39, once all parties have authenticated to one another, the data transmission begins. Master 1 sends data encrypted by Slave 2's public key as denoted by the "????" in the data packet, for which Slave 2 is the only device that can decrypt the data. Once the data transmission is complete, the bus operation will end, as is the case for any normal bus operation.

The benefit of using bus encryption is that it is an easy solution for the designer and manufacturer. Communities have come up with great encryption protocols. Ciphers such as RSA, Advanced Encryption Standards (AES), Data Encryption Standard (DES), etc. can be used to eliminate confidentiality vulnerabilities for IC bus policies. Also, using simple Message Digest 5 (MD5) cryptographic hashes together with a nonce at the end of every data exchange from sender and receiver will help ensure that there are no modifications during transmission. However, even though there are clear benefits to using encryption on bus protocols, there is some cost. First of all, the protocol is completely dependent on a good bus encryption algorithm, and it has to be implemented correctly in hardware. Attackers can exploit weak ciphers and bad hardware implementations. Cryptographic keys must be properly managed, and the encryption and decryption operations will consume resources of time, circuit area, and power, as discussed below. The performance of the cipher compared with the speed of the bus must be carefully considered when designing a bus that is integrated with an encryption mechanism. While a strong cipher addresses confidentiality vulnerabilities, other solutions for addressing integrity and availability vulnerabilities are needed.

As we have stated in earlier chapters, the primary objective of a bus security policy is to ensure availability through timely and reliable delivery. This solution will not eliminate any of the availability flaws that we have generated in Chapter IV. Additionally, using encryption will slow down the speed and efficiency of the bus protocol. For speeds of 100-400 kb/s associated with I2C, this might not make much of a difference, but for HyperTransport bus speeds of 25 Gigabytes/second (Gb/s), it may result in substantial performance degradation, depending on the efficiency, specifically the throughput, of the implementation of the encryption and decryption operations. Manufacturers and designers may not find this desirable enough to implement in their bus

protocols and may therefore choose speed and efficiency over security. If the designer chooses to use bus encryption it would have to be on a device where timing and speed are not a priority, and where some other measures are implemented to eliminate availability and integrity flaws.

F. SECURITY ENHANCED COMMUNICATION ARCHITECTURE (SECA)

SECA is not currently implemented in any fielded systems, but rather is a theoretical solution from NEC laboratories to address a SoC architecture's security concerns. SECA is to be embedded into a system for which it acts as an additional reference monitor to enforce the SoC's security policies [25].

As we can see from Figure 40, SECA is embedded in the AMBA SoC bus. SECA is a centralized module consisting of a single Security Enforcement Module (SEM) and a Security Enforcement Interface (SEI) for each slave device. SEM can exist as a master or slave and act as an enforcer of program security policies, access control, and intrusion detection. The SEI is placed on the slave devices and helps the SEM filter values that reach the data and control registers of peripherals [25].

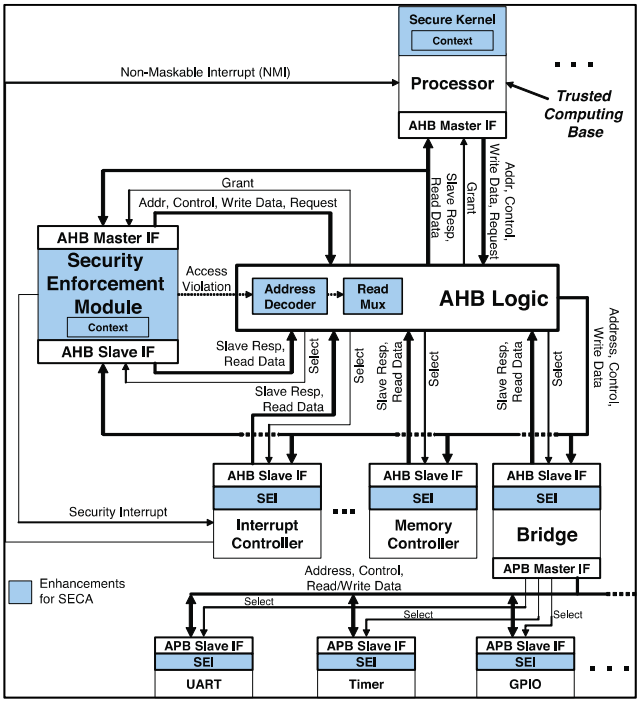

Figure 40. SECA on AMBA Architecture (From [25]).

SEM enforces system security through the Address Protection Unit (APU), Data-based Protection Unit (DPU), and Sequence-based Protection Unit (SPU). The APU enforces rules for read-write access control for each device. The APU utilizes a lookup table for each entry, which contains permissions. Using the DPU, memory space is set aside to store access levels to reduce time wasted on continuous address look-ups. The SPU relies on the basic sequences of transactions through the security of formal methods (i.e., automata). Security policies have associated formal methods, and the SPU ensures that information flows in accordance with those prescribed states [25].

Essentially, the SEM is an enhanced arbiter with SEI modules attached to each device reporting to the SEM on their individual device security state vice having one arbiter manage all devices. This is great idea for SoC buses such as AMBA, CoreConnect, and even Wishbone. If developed and implemented in the open market, this could address SoC bus confidentiality, integrity, and availability vulnerabilities; however, the performance overhead of using security schemes in SoC designs has not been fully addressed. Furthermore, this solution does not alleviate the problem for non-SoC buses; another solution for HyperTransport and I2C is needed.

G. TRUSTZONE

TrustZone is a product developed and sold by ARM Ltd. that can be used on most SoC buses. The intent of TrustZone is to provide a one-size-fits-all solution. TrustZone allows the SoC designer to divide the devices on the SoC and place them into two worlds. One world is secure, and the other is considered normal. Figure 41 is an abstraction TrustZone [26].

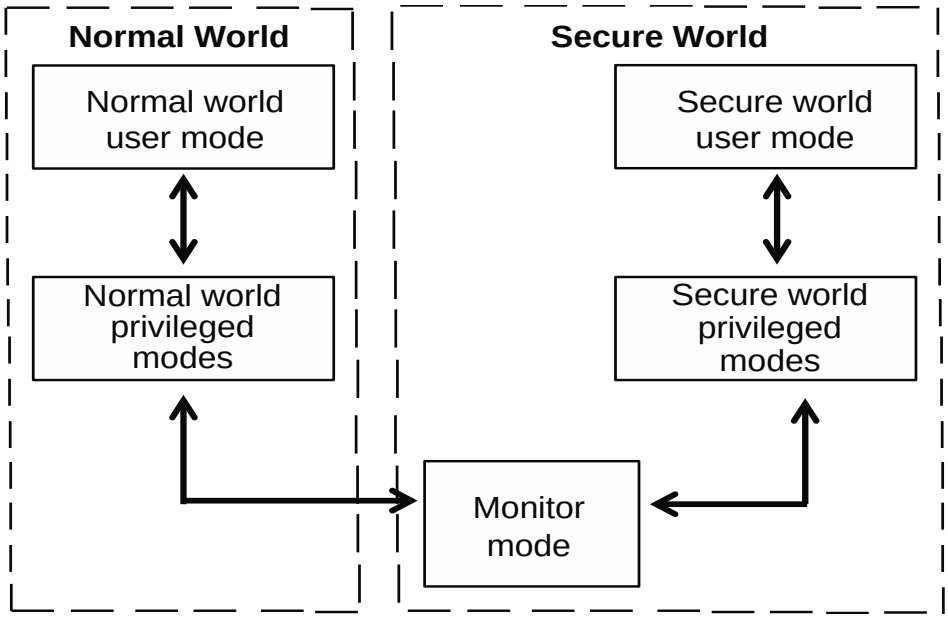

Figure 41. TrustZone topology (From [26]).

The objective of TrustZone is fairly straightforward. An environment has been constructed that protects the programmable environment from confidentiality and integrity attacks. The two worlds may communicate with one another through a Monitor mode, which is placed in the secure world. The processing time between both worlds is shared using time slices [26].

This can work for almost any SoC and provides flexibility. The process is relatively simple and easy to understand, but the overhead of using security schemes in SoC designs has not been fully addressed. TrustZone may be useful for SoC buses, but it will not help I2C and HyperTransport [26].

H. PARALELLIZED ENCYRPTION INTEGRITY CHECKING ENGINE (PE-ICE)

The purpose of using the PE-ICE solution for SoC is to protect data confidentiality and data integrity from MITM attacks. As seen in Figure 42, data confidentiality is protected using a block encryption algorithm such as Advanced Encryption Algorithm (AES). Data integrity is protected using a three-stage process. Once the encryption has been performed, each encrypted block depends on the corresponding plaintext blocks that follow. Second, the data is verified using a nonce that

was sent along with the encrypted data. Since a nonce is a one-time value, it hard to replicate. Furthermore, if there is an integrity violation, then PE-ICE raises the integrity checking flag for further investigation. Finally, the SoC is responsible for encrypting and decrypting. A Tag is associated with each block, and each SoC must have a preexisting knowledge of each Tag. If the Tag was modified by an attacker, then the processor will not be able to open the block and therefore know that it had been tampered with [27].

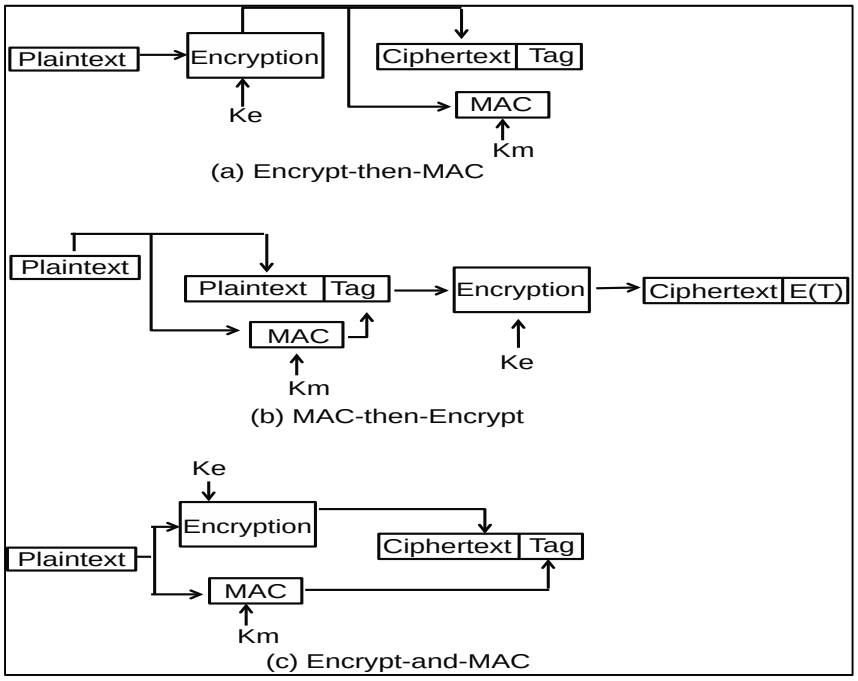

Figure 42. PE-ICE Process (From [27]).

PE-ICE is an innovative approach, but just like our general encryption solution, there is some performance loss. PE-ICE adds protection to the integrity of the data; however, if an attacker continues to modify the block, then the recipient will be continuously denied from accessing information. An availability solution is also needed.

I. GATE-LEVEL INFORMATION FLOW TRACKING (GLIFT)

As technology progresses, so does the sophistication of attacks. IC buses are added to circuits, which use logic gates that provide for fast and efficient information

flow. There has been considerable work in data-flow tracking architectures for detecting malicious behavior such as code injection attacks and cross-site scripting [28].

GLIFT is a proposed method for tracking logic gates. The idea behind GLIFT is that every gate has a shadow gate that processes the Tags. For example, an AND gate has two data inputs, and its shadow gate has two tag inputs. The AND gate computes the data output based on the traditional AND truth table, but the shadow gate computes the tag output based on a special truth table that considers both the data inputs and the tag inputs. A one-bit tag can represent that the corresponding data is either tainted or untainted. GLIFT can detect problems in circuits where tainted and untainted data must be kept separate. GLIFT will be able to identify that tainted data has propagated to a part of the circuit where only untainted data should reside. GLIFT shadow logic can be added to a circuit that is put into production, with the shadow logic operating in parallel with the data logic, or GLIFT can be used during the development phase only and removed once it is determined through testing that the circuit does not have improper information flow [28].

We argue that GLIFT could be applied to analyze bus circuitry, and it already has been applied in the laboratory for this purpose. However, GLIFT is still under development. GLIFT, together with other dynamic information flow tracking (DIFT) and tagging techniques, offers unique benefits for hardware-oriented security and trust.

J. ELIMINATION MATRIX

We will take all the techniques discussed in this chapter and match them to the IC bus protocols that we have discussed in this thesis. Table 7 depicts a matrix with all the flaw elimination techniques represented in the columns and the buses are represented in the rows. An “X” is marked to signify that the elimination technique could be applied to that particular bus. HyperTransport is represented by “HT,” Wishbone by “WB,” and CoreConnect by “CC.”

	Golden Model	Redesign	Round Robin	Arbiter Redesign	Encryption	SECA	TrustZone	PE-ICE	GLIFT
I2C	X		X		X				X
AMBA	X	X	X	X	X	X	X	X	X
HT	X				X			X	X
WB	X	X	X	X	X	X	X	X	X
CC	X	X	X	X	X	X	X	X	X

Table 7. Elimination matrix.

K. SUMMARY

This concludes our discussion of FHM penetration testing. We have discussed over 10 different methods and technological advances that can help protect IC bus security policies. Some methods are not practical, such as the Golden Model, whereas others impose performance degradation (i.e., encryption) or are not yet fully available for use in production-quality (i.e., sign-off) tools (e.g., SECA and GLIFT). Vendors and users of IC bus protocols have to make their own decisions on what techniques to use, but hopefully, our analysis will provide some insight for making these choices. Chapter VII will form conclusions and tie together the discussion from the earlier chapters, including the original hypothesis and thesis questions, what we discovered, and what remains to be done.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

A. INTRODUCTION

In this chapter, we summarize our results and compare what we expected to find against what we actually found (i.e., Do buses have any security relevant properties? Is one bus more secure than another?). In addition, we discuss opportunities for future work.

B. TAKEAWAYS

Our conclusions are based on our understanding of the similarities in IC bus operations, flaws found, and methods used to eliminate flaws with. Every bus roughly has the following operational sequences in common:

- A master device must request use of a slave device
- A reference monitor is in place in order to ensure the master device has access
- The reference monitor will also ensure that the slave device is free to use
- The reference monitor will grant access if steps 2 and 3 are met
- The master device will then begin its data transmission

Every type of IC bus is designed differently, whether it is for speed and efficiency or a particular platform. They are all called buses because they meet the standard definition of reliably delivering data from one device to another. We discovered through our process of FHM that these buses carried some striking similarities in flaw exploits, which will also lead to some similar principles in fixes and mitigations. We found that many of the exploits were based on corruptions introduced by designers, which ranged from flagrantly overlooked problems to ones that would have been difficult to know. All IC buses are built using gate logic, which is complex in itself, but adding bus protocols and arbitration sequences makes it even more difficult to be completely thorough in testing and analysis. Because of some lack of oversight or design missteps, attackers could make these kinds of flaws work to their advantage, typically in the form of a DoS. Additionally, we found that some of the flaws were attributable to designers choosing the wrong bus for their platform. There are a lot of IC buses from which to choose, ranging

from highly untrusted to highly trusted. A user could choose a highly trusted bus, such as CoreConnect, that is far too complicated for a simple SoC device, or they could choose an untrusted bus like I2C that is too simple for a system with complex hardware. Either way, this would result in some form of DoS because the bus would not be able to recognize certain commands from the devices.

Another common flaw that kept appearing through the various flaw generation processes was bus hogging. A lot of the IC bus devices allowed for priorities and/or the protocol never specified a maximum amount of time a device may use the bus at one time. These would result in bus hogging, and we were able to resolve this through suggesting schemes that use fixed time slices such as round robin schemes. This would resolve most of the availability flaws caused by bus hogging.

Additionally, we found that a lot of buses are designed with a shared architecture, which is exploitable through eavesdropping on another device's bus transmissions. To resolve confidentiality violations, we found that all IC buses can easily apply some form of cryptography to protect the data traversing the bus.

We were also able to generate a number of MITM attacks for all of the buses. These attacks would result in violations of confidentiality, integrity, and availability. We have addressed the common mitigations to confidentiality and availability, but integrity has proved to be the toughest. What seems to be the common solution to this is adding PE-ICE to check for altered data, or to use input signal wrappers that would alter gate logic so that only the participating master and slave device would be able to see the data on the bus.

C. WHAT WE KNOW NOW

In Chapter I, we asked whether IC bus protocols have any inherent security properties and if one bus is more secure than another. We initially assumed that buses do have inherent security properties, but only those associated with ensuring availability. Our research has lead us to the conclusion that yes, the IC buses do have security properties related to availability, but they additionally display some minor security properties that are related to availability and integrity. The use of reference monitoring

and an address decoder helps ensure that when a device accesses other devices, it does not read or write to devices it is not supposed to. This is a relatively minor security property, but we realize that more has to be done to ensure stronger confidentiality, integrity, and availability security properties.

The other question that we were able to resolve was whether one bus was more secure than the other. The answer is yes! Different buses display stronger security properties, reference monitor schemes, or have better control of their life-cycle development. In order to demonstrate our assertion, Table 8 is a list of all flaws generated from Chapter V, and it is used to compare all IC buses discussed in this thesis. This table is used to convey which bus is more secure than the other by demonstrating which flaws can apply to which particular buses, as denoted by the "X."

	I2C	AMBA	Hyper-Transport	Wishbone	CoreConnect
I2C MITM	X			X	
Covert Delivery	X			X	
Infinite Wait State	X			X	
Ignore	X	X	X	X	X
Oversimplifying	X	X	X	X	X
Unstable Power	X	X	X	X	X
Full Access	X	X	X	X	X
DoS	X	X	X	X	X
Bus Hogging		X	X	X	X
Read-write-deny MITM	X	X	X	X	X
P2P read-write deny			X	X	
Shared bus read-deny	X			X	
Crossbar read-deny	X			X	
Rearbitration		X		X	X
Large Latency		X		X	X
Request Termination		X		X	X
TOTAL:	**11**	**10**	**8**	**16**	**10**

Table 8. Bus security weaknesses.

All the generated flaws were analyzed to determine whether they were applicable to the bus and which ones are based on similar architecture, operational protocol, and reference monitoring schemes. We were able to determine that HyperTransport, despite

being considered to have relatively weak reference monitoring, displayed the greatest amount of security, followed by AMBA and CoreConnect. I2C and Wishbone rounded out the bottom two, as we initially hypothesized.

HyperTransport displayed stronger security property because it does not use a shared architecture, and data moves at such a high speed that it is difficult to read from or write to it without the slowing of the bus being noticed. HyperTransport comes from a single-source entity, which has a center of excellence, colloquia, white papers, and other forums dedicated to the design and efficiency of the bus.

AMBA and CoreConnect were rated the same, and it can be asserted that whatever flaw or vulnerability that AMBA has, then CoreConnect would have the same and vice versa. Both buses are very similar in architectural design and platform use. We can assert that CoreConnect is a little more secure because it has a reference monitor for each of the bus components (i.e., three reference monitors), whereas AMBA only has one. Both buses are products of single-source vendors that continually update their bus in order to be efficient and remain competitive in the marketplace.

It was assumed throughout this thesis that I2C was one of the least secure of the all the buses. Its shared architecture allows for every device attached to the bus to read whatever data is traversing it. Also, there are multiple sources of procurement, it has an outdated protocol that will not be updated, it has a weak arbitration scheme, and it is typically used on televisions and radios, which cause little motivation to incorporate stronger security measures into such a low-security application.

Wishbone can take on virtually any form. The design, architecture, protocol, and protection measures are left to the individual designer, which can make this bus one of the strongest or the weakest of all five. Additionally, Wishbone is the weaker protocol because the bus can be downloaded from open-source hardware sites, such as OpenCores, where the code is posted for anyone to download without any assurance of security or its compatibility with the user’s platform.

In order for a bus to be secure, it must have a strong architecture (i.e., one that is not shared), separation of trusted components from normal components, and strong

addressing schemes that are not subject to loose interpretation. Also, it should originate from trusted sources that offer detailed specifications, a manual in English, vendor assistance, and protocol analyzer support.

D. BIGGEST CHALLENGES

Throughout this thesis process, the author has faced many challenges—primarily, not having the physical or simulated buses to actually test flaws with. Everything had to be done through analytic research that was grounded in documentation or previous testing from other researchers. Having no experience in penetration testing, no background in hardware security, and no IC bus knowledge before starting this thesis proved challenging.

E. FUTURE WORK

This thesis has laid the foundation for future work in IC bus security testing. Ultimately, it is the author's goal to have someone with an electrical engineering or computer engineering background from a university to apply this thesis as the groundwork to test our ideas on FPGAs and then apply the suggested flaw elimination techniques.

However, that is a big leap from this thesis, and it needs to be taken in smaller steps by first developing our threats and exploits on software, followed by development of elimination techniques. Once those have been tested, then another thesis can move towards implementing them on an actual processor and measure loss or gain in efficiency and determine the optimal elimination techniques for IC bus protocols. Additionally, future work can also develop more advanced bus exploitations than those mentioned and discovered in this thesis.

F. SUMMARY

The author hopes that this thesis is able to provide valuable insights to the hardware security community in order to shed some light on the exploitation of bus protocols. Additionally, we were able to discover that buses do have individual security properties, and one can be more secure than the other. The next step is to use this thesis as a basis for further analysis and testing on software and hardware modules.

LIST OF REFERENCES

[1] L. Null and J. Lobur, *The essentials of computer organization and architecture*. Sudbury, MA: Jones and Bartlett Publishers, 2003, pp. 179–181.

[2] S. Pasricha and N. Dutt. (2008). *On-Chip communication architectures: System on chip interconnect* [Online]. Available: http://www.loc.gov/catdir/toc/ecip0810/2008004691.html

[3] B. Schaur. Multicore processors – A necessity. *ProQuest discovery guides* [Online], 2008, pp. 1–14. Available: www.csa.com/discoveryguides/multicore/review.pdf

[4] NIST. *Computer security resource center* [Online], November 12, 2011. Available: http://csrc.nist.gov

[5] A. Shostack. *Experiences threat modeling at Microsoft*, 2009 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.143.1410

[6] N. Potlapally, "Hardware security in practice: Challenges and opportunities," presented at the Hardware-Oriented Security and Trust (HOST), *2011 IEEE International Symp.*, San Diego, CA, 2011.

[7] J. D. Villasenor, "The hacker in your hardware," *Scientific American*, vol. 303, pp. 82–87, August, 2010.

[8] S. Adee, "The hunt for the kill switch," *IEEE Spectr.*, vol. 45, pp. 34–39, May, 2008. Available: http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch

[9] S. T. King et al., "Designing and implementing malicious hardware," presented at *Proc. 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, San Francisco, CA, 2008. Available: http://dl.acm.org/citation.cfm?id=1387709.1387714

[10] D. Paret, *The I2C Bus: From Theory to Practice.* Chichester, New York: John Wiley & Sons, 1997.

[11] AMBA Specification Revision 2.0, ARM, 1999.

[12] HyperTransport Consortium. [Online]. Available: http//:www.hypertransport.org

[13] OpenCores. *Wishbone B4: WISHBONE system-on-chip (SOC) interconnection architecture for portable IP cores*, *4*th ed. [Online]. (2011, November 5). Available: cdn.opencores.org/downloads/wbspec_b4.pdf

[14] IBM. *CoreConnect bus architecture.* [Online]. (2011, November 1). Available: https://www-01.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture

[15] C. Weissman, *Handbook for the Computer Security Certification of Trusted Systems*. Washington, D.C., 1995.

[16] Philips I2C Bus. [Online]. Available: http://www.i2c-bus.org

[17] J. W. Bruce et al., "Personal digital assistant (PDA) based I2C bus analysis," *IEEE Trans. on Consum. Electron.*, vol. 49, no. 4, pp. 1482–1487, 2003.

[18] Lok-Won Kim and J. D. Villasenor, "A system-on-chip bus architecture for thwarting integrated circuit trojan horses," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 10, pp. 19–21, 2011.

[19] Lok-Won Kim et al., "A trojan-resistant system-on-chip bus architecture," presented at the Military Communications Conference (MILCOM), 2009.

[20] A. Huang. (2003). *Hacking the Xbox: An introduction to reverse engineering* (Unlimited). [Online]. Available: http://www.loc.gov/catdir/toc/ecip045/2003013303.html

[21] D. Anderson et al., *HyperTransport System Architecture*. Reading, MA: Addison-Wesley, 2003.

[22] OpenCores. [Online]. Available: http://www.OpenCores.com.

[23] A. Goel and W. R. Lee, "Formal verification of an IBM CoreConnect processor local bus arbiter core," presented at *Proc. of the 37th Annual Design Automation Conference*, Los Angeles, CA, 2000. Available: http://doi.acm.org/10.1145/337292.337384

[24] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, p. 10, 2010.

[25] J. Coburn et al., "SECA: Security-enhanced communication architecture," presented at *Proc. of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pp. 78–89, New York, NY, USA, 2005. [Online]. Available: http://doi.acm.org/10.1145/1086297.1086308

[26] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," *Information Quarterly*, vol. 3, no. 4, pp. 18–19, 24, 2004.

[27] R. Elbaz et al., "A parallelized way to provide data encryption and integrity checking on a processor-memory bus," presented at Design Automation Conference, 43rd ACM/IEEE, San Francisco, CA, 2006.

[28] M. Tiwari et al., “Gate-level information-flow tracking for secure architectures,” *IEEE Micro*, vol. 30, no. 1, pp. 92–100, 2010.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California